Par NIDHAL JELASSI
nidhal.jelassi@fsegt.utm.tn

PROGRAMMATION WEB AVANCÉE

ANGULAR

Chapitre 8 : HTTP Client

OBJECTIFS

- Comprendre le design pattern Observable et son implémentation avec RxJs
- Appréhender le Module HTTPClientModule d'Angular
- Utiliser les différents services du module HTTPClientModule
- Comprendre le principe d'authentification via les tokens
- Utiliser les protecteurs de routes (les guards)
- Utiliser les Interceptors afin d'intercepter les requêtes Http
- Déployer votre application en production

HTTP

- Angular est un Framework FrontEnd
- Pas d'accès à la BD
- Pas de possibilité de persistance des données
- Pas de puissance permettant des traitements lourds.

Solution :

- Le Moule HTTPClient

PROGRAMMATION ASYNCHRONE

- Ce sont des objets qui représente une complétion ou l'échec d'une opération asynchrone.

(https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Utiliser_les_promesses)

- Le fonctionnement des promesses est le suivant :
  - On crée notre fonction qui retourne une promesse
  - La promesse va toujours retourner deux résultats :
    - **resolve** en cas de succès
    - **reject** en cas d'erreur
- Vous devrez donc gérer les deux cas afin de créer votre traitement

EXEMPLE PROMISE

```
var promise2 = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve(3);
    }, 5000);
});

promise2.then(
    function (x) {
        console.log('resolved with value :', x);
    }
.catch( console.log('Erreur avec la Promesse');
)
```

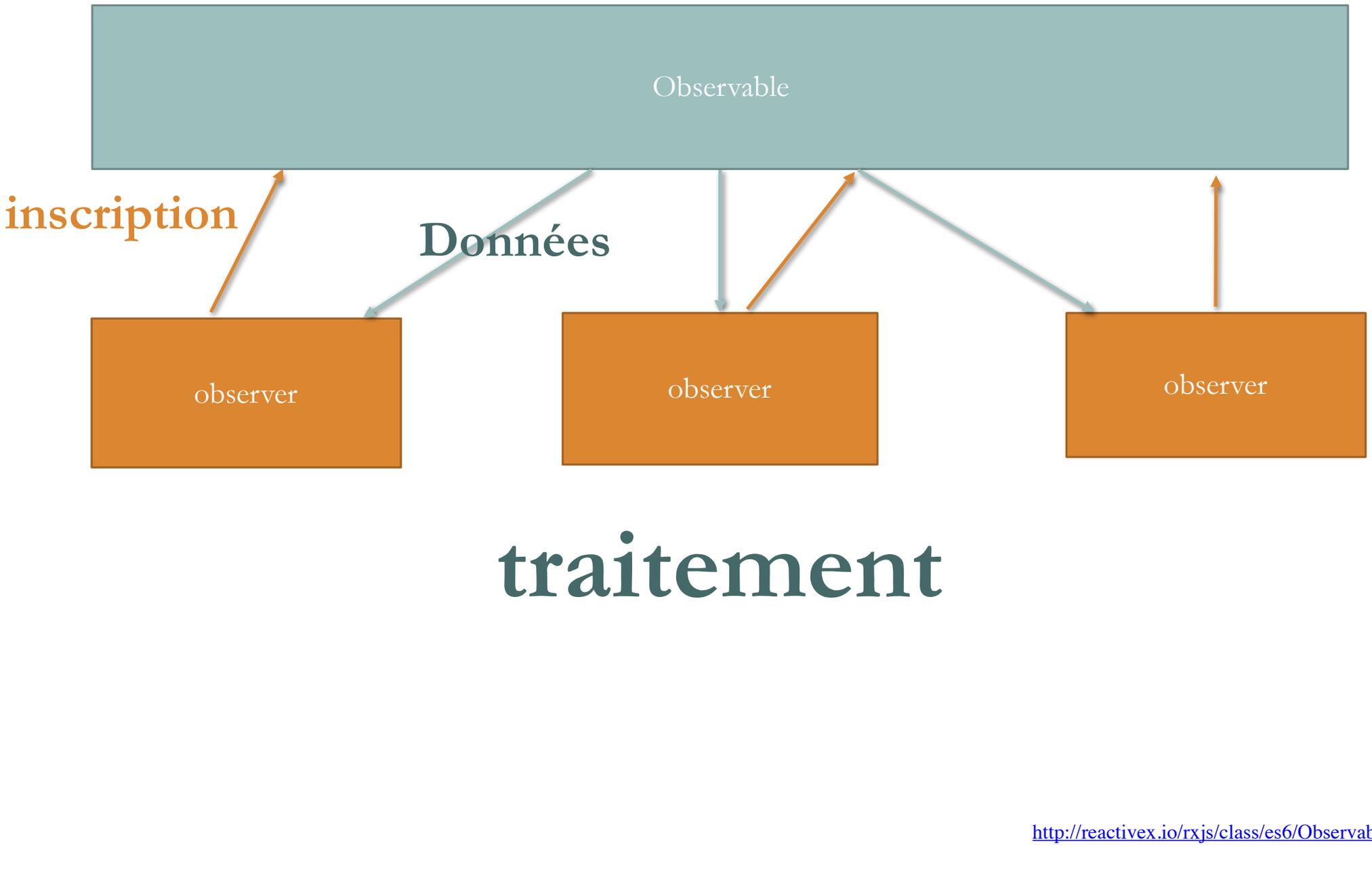
PROGRAMMATION RÉACTIVE

- Nouvelle manière d'appréhender les appels asynchrones
- Programmation avec des flux de données asynchrones

Programmation reactive =

Flux de données (observable) + **écouteurs d'événements(observer).**

OBSERVABLES ET OBSERVERS



http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html

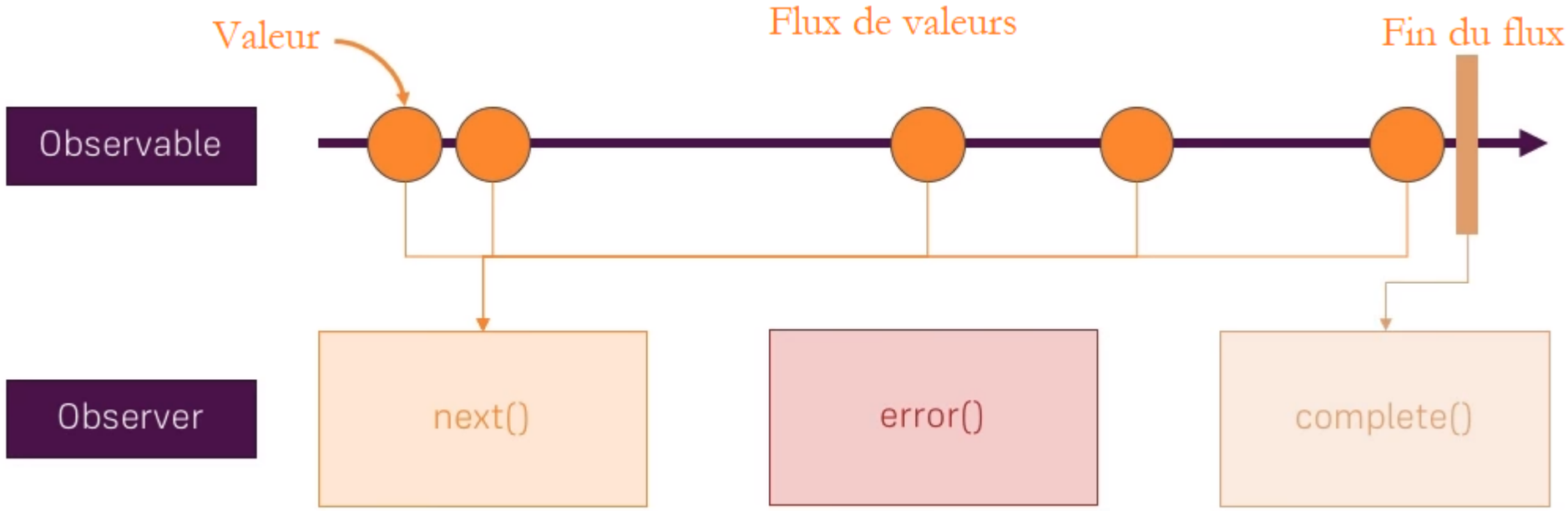
OBSERVABLES

- Le pattern design (patron de conception) Observable permet à un objet de garder la trace d'autres objets, intéressés par l'état de ce dernier.

- Il définit une relation entre objets de type un-à-plusieurs.

- Lorsque l’état de cet objet change, il notifie ces observateurs.

FONCTIONNEMENT



EXEMPLE OBSERVABLE

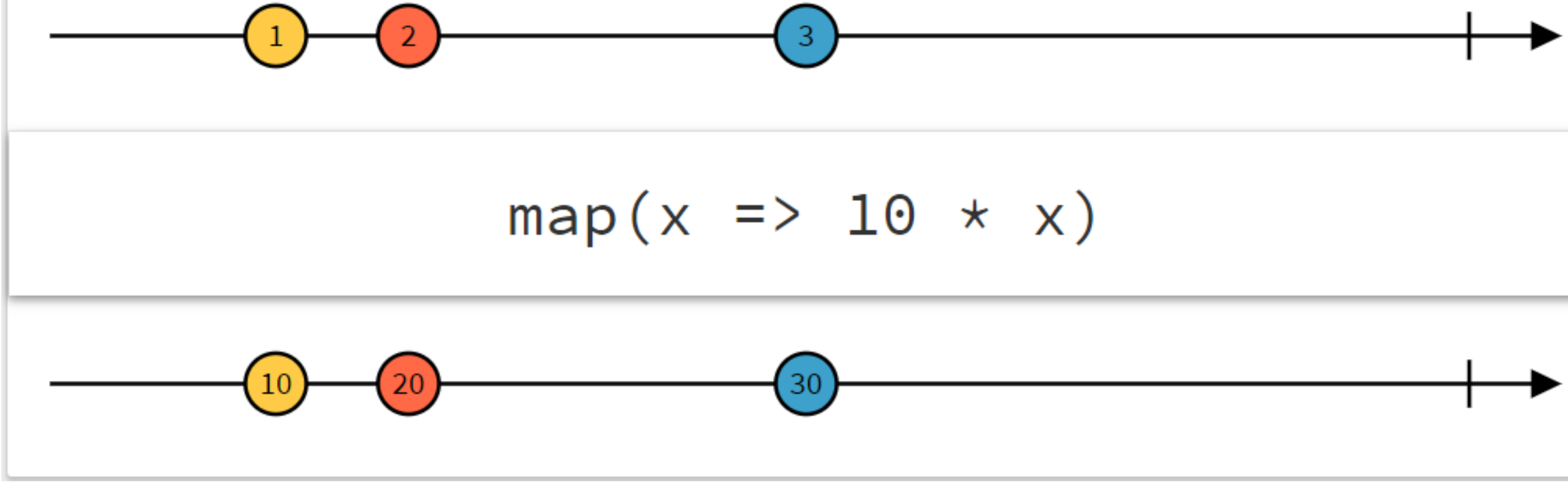
```
const observable = new Observable(
  (observer) => {
    let i = 5;
    setInterval(() => {
      if (!i) {
        observer.complete();
      }
      observer.next(i--);
    }, 1000);
  });
observable.subscribe(
  (val) => {
    console.log(val);
  }
);
```

APPLICATION

- Ecrire un composant qui affiche une suite d'images non stop.
- Utiliser un observable comme la source des images.

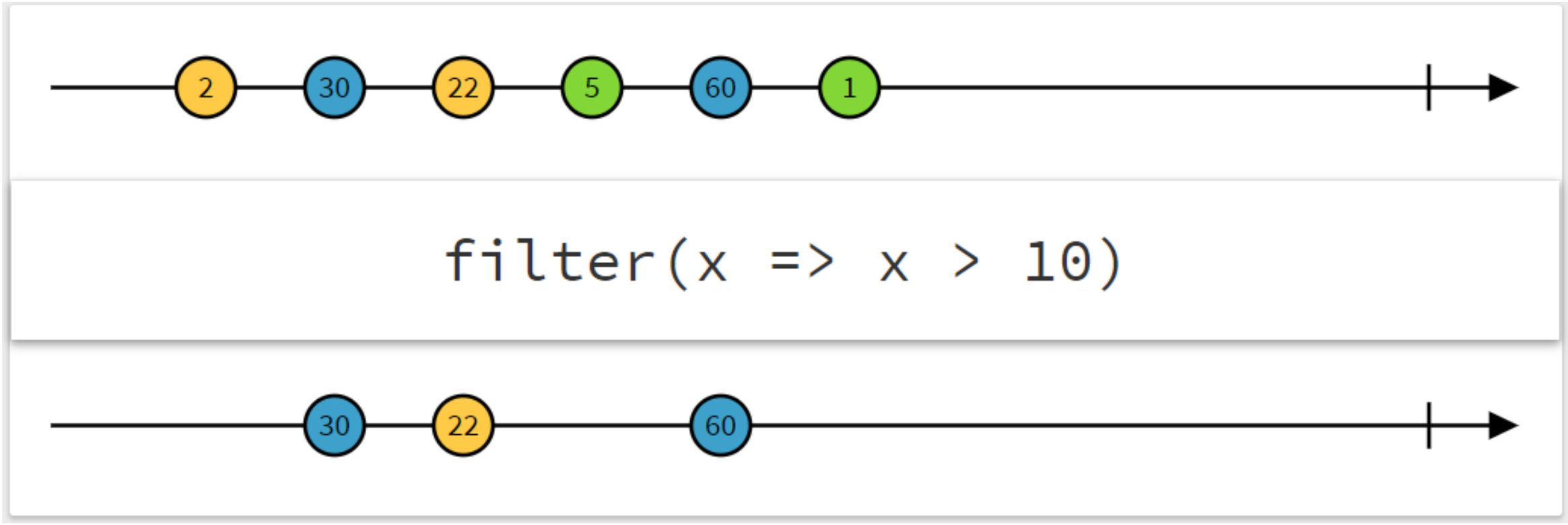
OPÉRATEURS UTILES AVEC LES OBSERVABLES

- Map



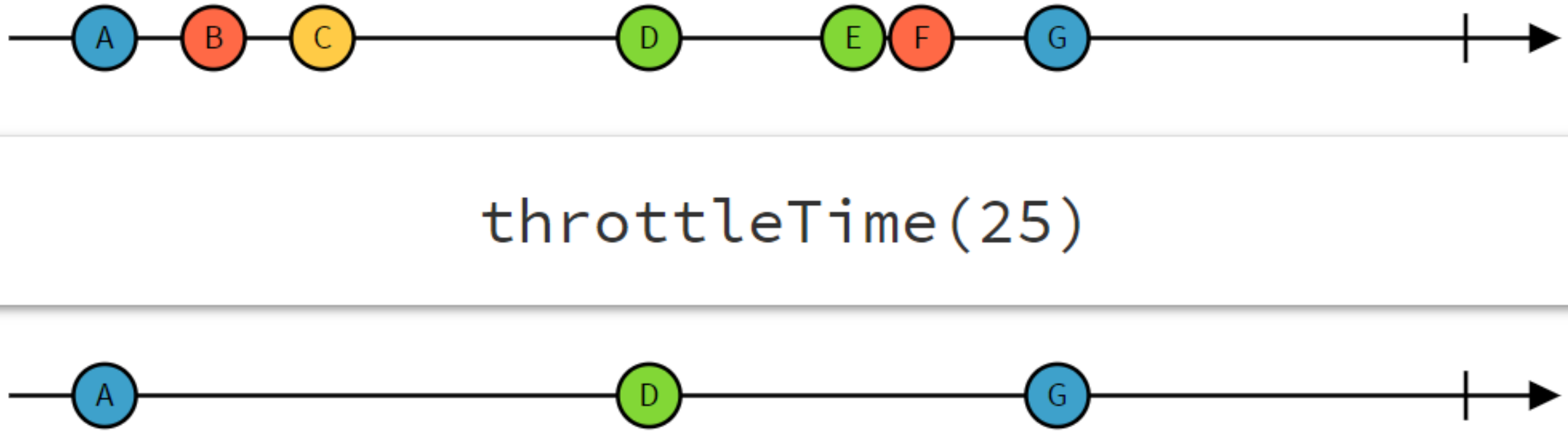
OPÉRATEURS UTILES AVEC LES OBSERVABLES

- Filter



OPÉRATEURS UTILES AVEC LES OBSERVABLES

- throttleTime



AUTRES OPÉRATEURS UTILISE

- https://angular.io/guide/rx-library

- http://reactivex.io/rxjs/manual/overview.html#operators

- http://rxmarbles.com/

INSTALLATION DE HTTP

- Le module permettant la consommation d'API externe s'appelle le HTTP MODULE.
- Afin d'utiliser le module HTTP, il faut l'importer de @angular/common/http (@angular/http dans les anciennes versions)
- Il faudra aussi l'ajouter dans le fichier module.ts dans le tableau d'imports.

```
import {HttpClientModule} from "@angular/common/http";
```

```
imports: [
  BrowserModule,
  FormsModule,
  HttpClientModule,
],
```

- Afin d'utiliser le module HTTP, il faut l'injecter dans le composant ou le service dans lequel vous voulez l'utiliser.

```
constructor(private http:HttpClient) { }
```

INTERAGIR AVEC UNE API GET REQUEST

- Afin d'exécuter une requête **get** le module http nous offre une méthode **get**.
- Cette méthode retourne un Observable.
- Cette observable a 3 callback function comme paramètres.
  - Une en cas de réponse
  - Une en cas d'erreur
  - La troisième en cas de fin du flux de réponse.

INTERAGIR AVEC UNE API GET REQUEST

```
this.http.get(API_URL).subscribe(
   (response:Response)=>{
      //ToDo with DATA
      },
(err:Error)=>{
   //ToDo with error
},
   () => {
      console.log('Data transmission complete');
}
);
```

- Afin d'exécuter une requête POST le module http nous offre une méthode post.
- Cette méthode retourne un Observable.
- Diffère de la méthode get avec un attribut supplémentaire : body
- Cette observable a 3 **callback functions** comme paramètres.
  - Une en cas de réponse
  - Une en cas d'erreur
  - La troisième en cas de fin du flux de réponse.

```
this.http.post(API_URL,dataToSend).subscribe(
  (response:Response)=>{
    //ToDo with response
    },
(err:Error)=>{
   //ToDo with error
},
  () => {
    console.log('complete');
}
);
```

PLUS D'INFOS

- Pour plus d'informations concernant ce module, vous pouvez vous référer à la documentation officielle d'Angular :

https://angular.io/guide/http

APPLICATION

- Accéder au site https://jsonplaceholder.typicode.com/

- Utiliser l'API des posts pour afficher la liste des posts. En attendant le chargement des données afficher un message « loading… ».

- Ajouter un input. A chaque fois que vous écrivez un élément dans cet input il sera ajouté dans la liste.

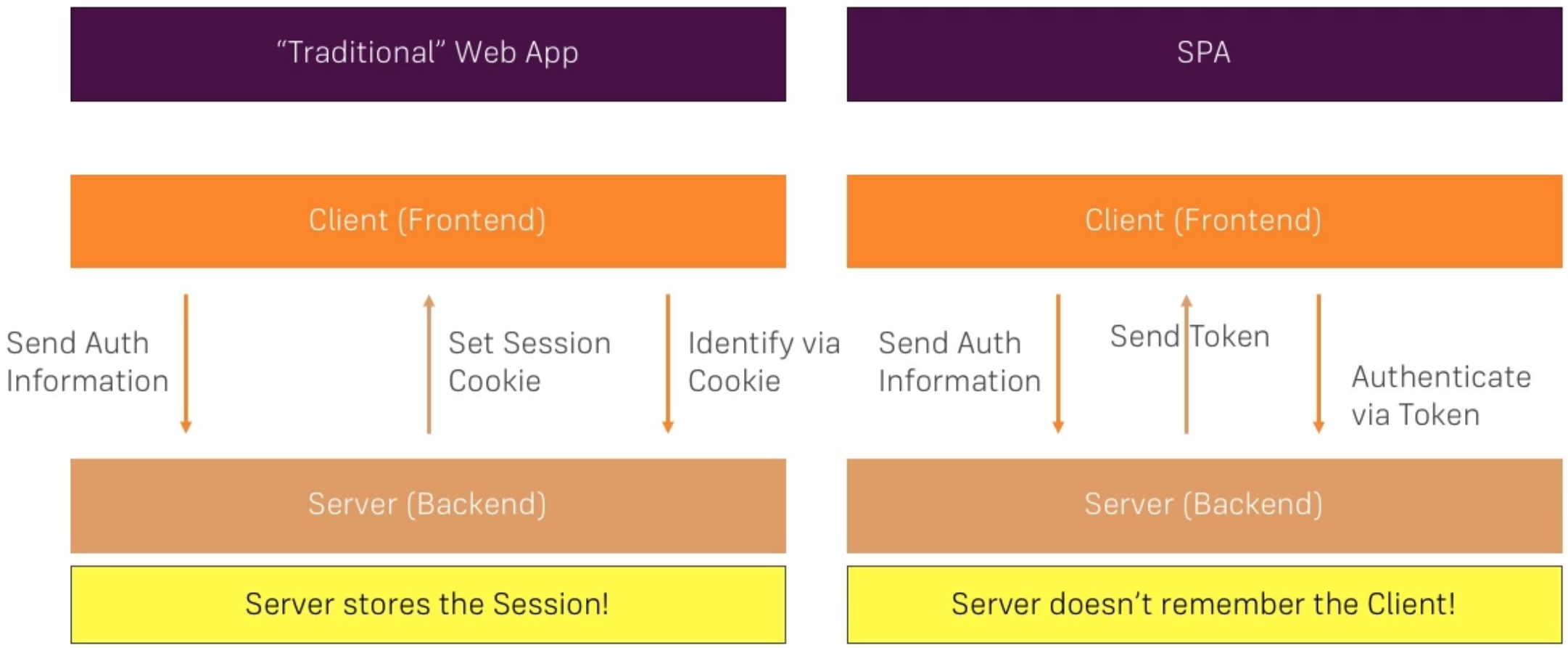
HTTPPARAMS

- Afin d'ajouter des paramètres à vos requêtes, le HttpClient vous offre la classe HttpParams.
- Cette classe est une classe immutable (read Only).
- Elle propose une panoplie de méthode helpers permettant de la manipuler.
- set(clé,valeur) permet d'ajouter des headers. Elle écrase les anciennes valeurs.
- append(clé,valeur) concatène de nouveaux headers.

- Toutes les méthodes de modification retourne un HttpParams permettant un chainage d'appel.

AUTHENTIFICATION

How does Authentication work?



LOOPBACK

- Loopback offre un mécanisme d''authentification prêt à l'emploi.
- Avec l'api de la classe user il vous permet d'ajouter des users et de vous connectez. Il permet aussi avec son module Loopback acl de créer des restrictions sur l'api.
- Une fois vos uri protégée, vous devez vous connecter pour pouvoir les utiliser.
- En vous connectant, il vous offrira un token. Vous devez l'utiliser à chaque appel de votre api.

INSTALLONS LOOPBACK

- Nous allons installer à présent le framework Loopback sur notre machine : npm install -g loopback-cli
- Vérifier votre version avec lb –v
- Créer votre première application avec la commande lb
- Renseigner le nom de votre projet
- Renseigner le type de votre projet

CRÉATION D'UN MODÈLE

- Créer votre modèle avec lb model
- Suivez les étapes, ajouter votre modèle et ajouter les champs qui le compose.
- Dans notre exemple nous allons utiliser un model « Personne »

```
? Entrer le nom du modèle : personne
? Sélectionner la source de données à laquelle associer personne : (n
o datasource)
? Sélectionner la classe de base du modèle PersistedModel
? Exposer personne via l'API REST ? Yes
? Forme plurielle personnalisée (utilisée pour générer l'URL REST) :
? Modèle commun ou serveur uniquement ? commun
```

MODÈLE OBTENU

- Dans ce cas nous allons utiliser une base de données MySql avec un id en auto-increment.

```
{
  "name": "personne",
  "base": "PersistedModel",
  "idInjection": true,
  "options": {
    "validateUpsert": true
  },
  "properties": {
    "cin": {
      "type": "number",
      "required": true
    },
    "name": {
      "type": "string",
      "required": true
    },
    "firstname": {
      "type": "string",
      "required": true
    },
    "age": {
      "type": "number",
      "required": true
    },
    "quote": {
      "type": "string",
      "required": true
    },
    "metier": {
      "type": "string",
      "required": true
    }
  },
```

ASSOCIER UNE BD MYSQL

- Installer le mysql-connector
- npm install loopback-connector-mysql --save
- Configurer votre datasource dans le fichier datasource.json et ajouter la configuration de votre base de données mySql.
- Associer la datasource à votre modèle dans le fichier model-config.json.

ASSOCIER UNE BD MYSQL

```
{
  "db": {
    "name": "db",
    "connector": "memory"
  },
  "personneDb": {
    "host": "localhost",
    "port": 3306,
    "url": "",
    "database":
"test_pdo",
    "password": "",
    "name": "personneDb",
    "user": "root",
    "connector": "mysql"
  }
}
```

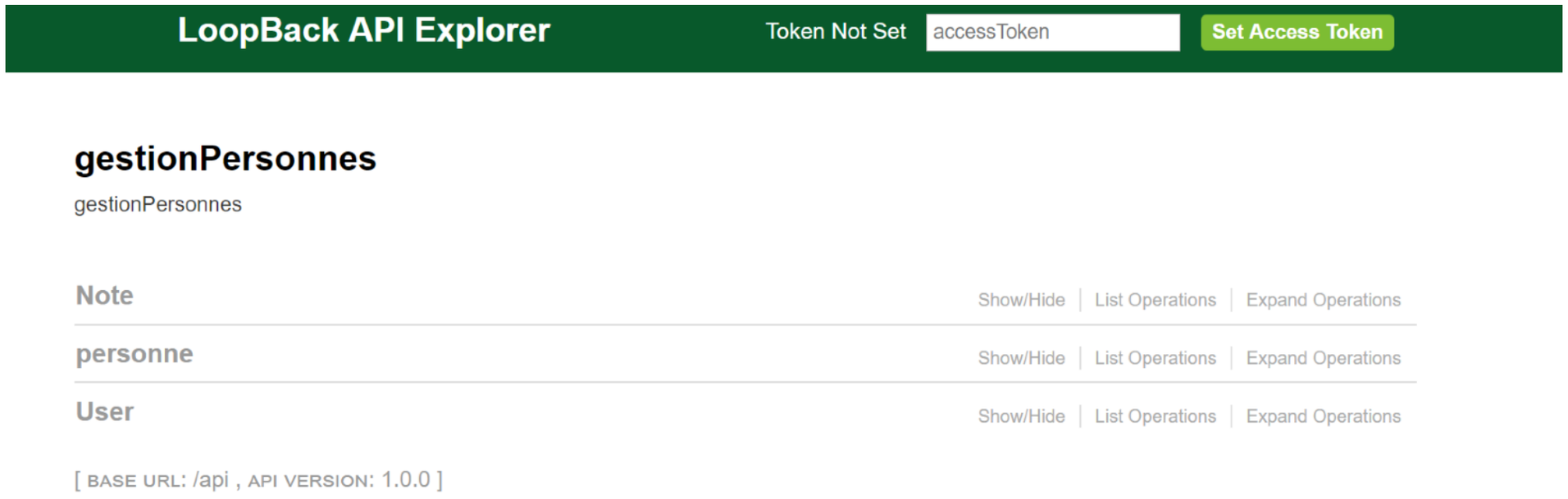
datasource.json

```
"Note": {
  "dataSource": "db"
},
"personne": {
  "dataSource":
"personneDb",
  "public": true
}
```

Model-config.json

LANCER VOTRE API

- Pour lancer votre application exécuter la commande node .
- Accéder à votre swagger en utilisant l'adresse http://localhost:3000/explorer
- Tester votre api.



RETOUR AU PROJET…

- Associer votre application à votre API Loopback. Tester les fonctionnalités d'ajout de suppression et de modification et de sélection de l'ensemble des personnes.
- Sachant que pour sélectionner une personne dont le nom contient une chaine donnée, loopback utilise la syntaxe suivante :

```
{"where":{"name":{"like":"%${name}%"}}}
```

- Ceci doit être fourni dans les paramètres de votre requête avec la clé filter.
- Ajouter un champ input. A chaque caractère saisie, la liste des choix doit automatiquement changer et n'afficher que les cvs qui contiennent la chaine saisie. En sélectionnant un des choix, rediriger l'utilisateur vers les détails du cv sélectionné.

AJOUTER UNE ACL AVEC LOOPBACK

- Nous voulons ajouter des contraintes d'accès à notre application.
- Nous voulons que seul les utilisateurs connectés puissent ajouter et supprimer des personnes.
- Ajouter les ACL en utilisant la commande lb acl.

```
? Sélectionner le modèle auquel a
ppliquer l'entrée ACL : personne
? Sélectionner la portée ACL : To
utes les méthodes et propriétés
? Sélectionner le type d'accès :
Ecrire
? Sélectionner le rôle Tous les u
tilisateurs non authentifiés
? Sélectionner le droit à appliqu
er Refuser explicitement l'accès
```

AJOUTER UNE ACL AVEC LOOPBACK

- Notre modèle s'en trouve ainsi mis à jour de cette manière :

```
"acls": [
   {
      "accessType": "WRITE",
      "principalType": "ROLE",
      "principalId":
"$unauthenticated",
      "permission": "DENY"
   }
```

APPLICATION

- Essayer d'ajouter une personne avec une requête de type POST. Utiliser directement avec le swagger de Loopback

- Que remarquer vous ?

AJOUTER LE TOKEN DANS LA REQUÊTE (1)

- Si la ressource demandé est contrôlé avec un token, vous devez y insérer le token afin d'être authentifié au niveau du serveur.

- Pour ajouter un token vous pouvez le faire via un objet HttpParams. Cet objet possède une méthode set à laquelle on passe le nom du token 'access_token' suivi du token.

- Vous devez ensuite l'ajouter comme paramètre à votre requête.

```
const params = new HttpParams()
  .set('access_token', localStorage.getItem('token'));
return this.http.post(this.apiUrl, personne, {params});
```

AJOUTER LE TOKEN DANS LA REQUÊTE (2)

- Une seconde méthode consiste à ajouter dans le header de la requête avec comme name ‘Authorization’ et comme valeur ‘bearer’ à laquelle on concatène le Token.

- Pour se faire, créer un objet de type HttpHeaders.

- Utiliser sa méthode append afin d’y ajouter ses paramètres.

- Ajouter la à la requête.

```
const headers = new HttpHeaders();
headers.append('Authorization', 'Bearer ${token}');
return this.http.post(this.apiUrl, personne,
{headers});
```

GUARDS

- Dans vos applications, certaines routes ne doivent être accessibles que si vous êtes authentifié. Ce cas d'utilisation se répète souvent et s'est un sous cas de la sécurisation de vos routes.

- Angular a pris en considération ce cas en fournissant un mécanisme via l'utilisation des Guard.

GUARDS

- Ce sont des classes qui permettent de gérer l'accès à vos routes.
- Un guard informe sur la validité ou non de la continuation du process de navigation en retournant un booléen, une promesse d'un booléen ou un observable d'un booléen.
- Le routeur supporte plusieurs types de guards, par exemple :
  - CanActivate permettre ou non l'accès à une route.
  - CanActivateChild permettre ou non l'accès aux routes filles.
  - CanDeactivate permettre ou non la sortie de la route.

CANACTIVATE

- Afin d'utiliser le guard canActivate (de même pour les autres), vous devez :

1. Créer un classe qui implémente l'interface CanActivate et donc qui doit implémenter la méthode canActivate de sorte qu'elle retourne un booléen permettant ainsi l'accès ou non à la route cible.

2. Vous devez ensuite ajouter cette classe dans le provider.

3. Finalement pour l'appliquer à une route, ajouter la dans la propriété canActivate. Cette propriété prend un tableau de guard. Elle ne laissera l'accès à la route que si la totalité des guard retourne true.

ETAPE 1

```
import { Injectable } from '@angular/core';
import {ActivatedRouteSnapshot, CanActivate, RouterStateSnapshot} from '@angular/router';
import {Observable} from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor() {
  }
  // route contient la route appelé
  // state contiendra la futur état du routeur de l'application qui devra passer la
 validation du guard

   canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):
 Observable<boolean> | Promise<boolean> | boolean {
     if (condition) {
       return true;
     }
     return false;
   }
 }
```

https://vsavkin.com/routeur-angular-comprendre-l%C3%A9tat-du-routeur-5e15e729a6df

ETAPE 2

```
providers: [
   TodoService,
   CvService,
   LoginService,
   AuthGuard,
 ],
```

App.module.ts

ETAPE 3

```
{
  path: 'color',
  component: ColorComponent,
  canActivate: [AuthGuard]
},
```

RETOUR AU PROJET…

- Ajouter les guards nécessaire afin de sécuriser vos routes.

- Faites en sorte qu’une personne déjà connectée ne peut pas accéder au composant de login.

LES INTERCEPTEURS

- A chaque fois que nous avons une requête à laquelle nous devons ajouter le token, nous devons refaire toujours le même travail.

- Pourquoi ne pas intercepter les requêtes HTTP et leur associer le token s'il est la à chaque fois ?

- Un intercepteur Angular (fournit par le client HTTP) va nous permettre d'intercepter une requête à l'entrée et à la sortie de l'application.

- Un intercepteur est une classe qui implémente l'interface HttpInterceptor.

- En implémentant cette interface, chaque intercepteur va devoir implémenter la méthode intercept.

MODIFIER LA REQUÊTE

- Par défaut la requête est immutable, on ne peut pas la changer.
- Solution : la cloner, changer les headers du clone et le renvoyer.

```
export const
AuthentificationInterceptorProvider = {
  provide: HTTP_INTERCEPTORS,
  useClass:AuthentificationInterceptor,
  multi: true,
};
```

```
providers: [
AuthentificationInterceptorPr
ovider
],
```

CLONER LA REQUÊTE

```
const newReq = req.clone({
    headers: new HttpHeaders()
// On peut ajouter des headers, des params …
  });
// Chainer la nouvelle requete avec next.handle
return next.handle(newReq);
```

DERNIÈRE ÉTAPE DU PROJET

- Créer un intercepteur qui injecte un token à chaque requête.
- Si le token n'existe pas, l'utilisateur ne peut rien faire des opérations déjà développés.

DÉPLOIEMENT

- Afin de déployé votre application, il vous suffit d'utiliser la commande suivante :
- ng build --prod
- Un dossier dist sera créer contenant votre projet
- Pour tester localement votre projet, télécharger un serveur HTTP virtuel avec la commande suivante :
- Npm install http-server –g
- Lancer maintenant votre projet à l'aide de cette commande :
- http-server dist/NomDeVotreProjet

DÉPLOIEMENT

- Créer un repository et mettez y votre projet

```
"deploy": {
    "builder": "angular-cli-ghpages:deploy",
}
```

- installer angular-cli-ghpages : ng add angular-cli-ghpages
- Ajouter cette configuration dans votre fichier angular.json
- Vérifier que vous avez déjà effectuer le build de votre applicaction avec ng build --prod
- Lancer command ng build --prod --base-href https://USERNAME.github.io/REPOSITORY_NAME/
- Lancer la commande ng deploy --base-href=/the-repositoryname/
- Accéder à votre page https://USERNAME.github.io/REPOSITORY_NAME
- En cas de mise à jour, relancer le même code.

DÉPLOIEMENT

- Ajouter cette configuration dans votre fichier angular.json et utiliser uniquement ng deploy :

```
"deploy": {
  "builder": "angular-cli-ghpages:deploy",
  "options": {
    "baseHref": "https://username.github.io/repoName/",
   }
}
```