Enhance Web Development

Angular 2

Communication serveur

- Le module HTTP
- Observable
- Observable.toPromise
- Gestion d'erreur
- Header

Ressources

- angular.io - HTTP Client
  https://angular.io/docs/ts/latest/guide/server-communication.html
- angular.io - Tutorial HTTP
  https://angular.io/docs/ts/latest/tutorial/toh-pt6.html
- Fake server
  http://jsonplaceholder.typicode.com/

Le module HTTP

Le module HTTP

- Module angular : *@angular/http*
- Importer dans le bootstrap de l'application

```
import { bootstrap }      from '@angular/platform-browser-dynamic';
import { HTTP_PROVIDERS } from '@angular/http';

import { AppComponent }   from './app.component';

bootstrap(AppComponent, [ HTTP_PROVIDERS ]);
```

- Simplifie l'usage des APIs natives XHR & JSONP pour utiliser les verbes de type REST
- Retourne des Observables (et optionellement des Promises)

Les méthodes disponibles

- Verbes HTTP REST

  http://www.restapitutorial.com/lessons/httpmethods.html

- Liste des méthodes

```
request() // [X] requête X, X étant un verbe HTTP
get() // [read] requête GET
post() // [create] requête POST
put() // [update/replace] requête PUT
delete() // [delete] requête DELETE
patch() // [update/modify] requête PATCH
head() // [head] requête HEAD
```

```
loadItems() {
  return this._httpService.get(BASE_URL)
    .map(res => res.json())
    .toPromise();
}
```

```
createItem(item: Item) {
  return this._httpService.post(BASE_URL, JSON.stringify(item), HEADER)
    .map(res => res.json())
    .toPromise();
}
```

Méthodes HTTP

```
updateItem(item: Item) {
  return this._httpService.put(`${BASE_URL}${item.id}`, JSON.stringify(item), HEADER)
    .map(res => res.json())
    .toPromise();
}
```

```
deleteItem(item: Item) {
  return this._httpService.delete(BASE_URL, HEADER)
    .map(res => res.json())
    .toPromise();
}
```

Méthodes HTTP

Les Observables

Qu'est ce qu'un observable

- Un flux d'évènement poussé dans le temps (lazy event stream) qui peut émettre zéro ou plusieurs évènements
- Composé de *subjets* et d'*observers*
- Le *subject* applique de la logique au flux et notifie l'*observer* quand il est nécessaire

Observable vs Promise

- Les observables sont retardés (lazy) - elle ne tourne pas tant que l'on a pas souscrit à ; contrairement au promise qui tourne de toute façon
- On peut annuler un observable ; une promise s'exécutera de toute façon
- On peut répéter un Observable (dans le cas d'une erreur réseau par exemple) avec la méthode *retry()*

Observable.subscribe

- On finalise un observable en souscrivant à ce dernier
- La méthode de subscribe supporte trois méthodes :
  - **onNext** : Quand des data arrivent
  - **onError** : Quand une erreur est levée
  - **onComplete** : Quand le stream est complet

```
// Creation d'un observable
this._httpService.get('http://domain.com')

// Consommer un Observable
source.subscribe(
  data => console.log(`Next: ${data}`),
  err => console.log(`Error: ${err}`),
  () => console.log('Completed')
);
```

Observable.subscribe

Observable.toPromise

- Le module @angular.http permet de transformer un Observable en Promise avec la méthode **toPromise**
- Permets de s'intégrer dans du code existant écrit pour

AngularJS en mode promise

```
// Creation d'un observable
this._httpService.get('http://domain.com')
  .toPromise()
  .then(res => res.json().data)
  .then(users => this.users = users)
  .catch(this.handleError)
```

Gestion d'erreur

- On doit **toujours** gérer les erreurs
- Utilisation de la méthode **.catch** qui prend l'erreur en paramètre

```
// Creation d'un observable
this._httpService.get('http://domain.com')
  .catch(error => {
    console.log(`An error occurred ${error}`)
    return Observable.throw(error.json().error || 'Server error')
  })
```

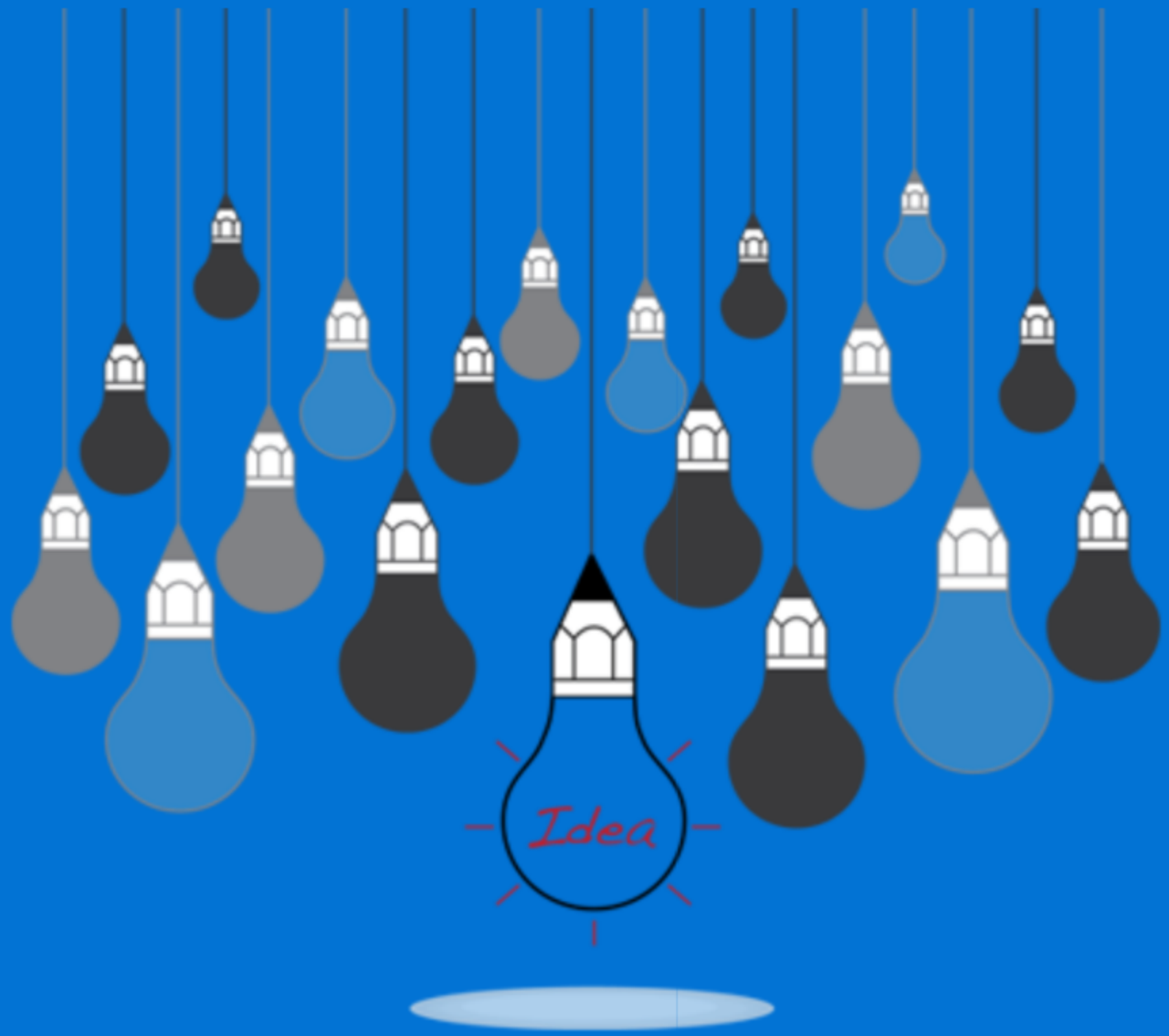
Headers

- Les Headers sont gérés avec les imports **Headers** et **RequestOptions** du paquet **@angular/http**
- Les headers permettent de personnaliser une requête

https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

```
let headers = new Headers({'Content-Type': 'application/json'})
let options = new RequestOptions({headers: headers})
this._httpService.post('users.json', '{}', options)
```

Demo time !



Défi

- Créer une méthode dans le service **widget** qui récupère des données via un service HTTP
- Souscrire à cette méthode dans un composant et afficher les données dans le template
- Effectuer la même méthode dans le service **widget** en utilisant des promesses
- BONUS : S'amuser avec d'autres verbe HTTP (post/put/delete)