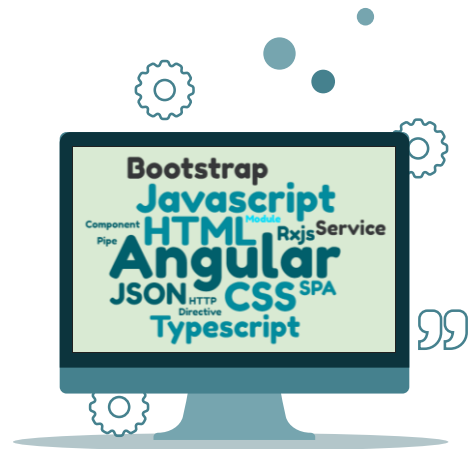


## 08

## Communication Client - serveur



### Chapitre 8 Communication Client-Serveur

01

HttpClient

03

Atelier 5

02

Appeler un Backend

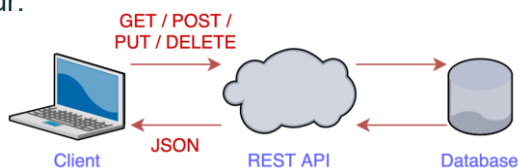
01

# HttpClient



## API REST

- Une **API REST** (Representational State Transfer) est une interface qui permet la communication entre un client (**frontend**) et un serveur (**backend**) via le protocole **HTTP**.
- Elle repose sur le principe des requêtes et réponses (**HTTP Request - HTTP Response**) et utilise différentes méthodes HTTP, telles que **GET**, **POST**, **DELETE** et **PUT**, pour échanger des données.
- Ces méthodes servent à effectuer diverses opérations, comme la **récupération**, la **création**, la **suppression** ou la **mise à jour** de **ressources** sur le serveur.



©MAD

mohamedanouer.dahdeh@fsb.ucar.tn

## Angular & les serveurs

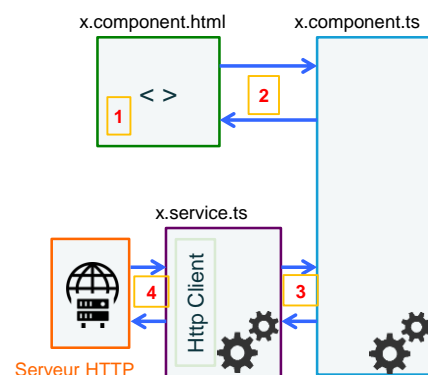
- Angular possède le module **HttpClientModule** qui facilite la réalisation de requête HTTP vers n'importe quel serveur
- Pour les tests :
  - **json-serveur** est un serveur HTTP open-source pour configurer rapidement une API REST en utilisant un fichier JSON comme source de données. Cela peut être utile à des fins de développement et de test lorsque les développeurs frontend n'ont pas de serveur backend complet disponible.
  - Installation : `npm install -g json-server@0.17.0`
  - Création (dans le répertoire du projet) d'un fichier `mydb.json` qui servira de BD
  - Démarrage du serveur : `json-server -p 3000 mydb.json`
  - Accès à une ressource : `http://localhost:3000/<nom_ressource>`

©MAD

mohamedanouer.dahdeh@fsb.ucar.tn

## Principe

1. Les données sont saisies (ou affichées) dans le Template `.component.html`
2. La classe `.component.ts` peut récupérer des données dont la source est le Template `.component.html` pour les passer au service ou récupérer des données dont la source est le service pour les passer au template `.component.html`.
3. En faisant une injection de dépendance du service `.service.ts` dans `component.ts`, ce dernier peut l'utiliser pour persister ou récupérer des données
4. En faisant une injection de dépendance de la classe `HttpClient` dans `service.ts`, ce dernier peut effectuer des requêtes HTTP en précisant chaque fois la méthode et l'URL.



©MAD

mohamedanouer.dahdeh@fsb.ucar.tn

02

Appeler un Backend



Exercice 1 : Communication client-serveur

- Récupérer la liste des contacts à partir d'une API REST (json-server)
- Sauvegarder tous les changements dans une API REST
  - Ajout et Suppression

Edit Contact Component

Add Contact

Name

Sergey Brin

Email

sergey.brin@gmail.com

Website

http://google.com

Projects (separated with comma";")

Google, Alphabet Inc

Save

Add contact

Contacts Component

Contact list		
Name		
Doug Cutting	Detail	Delete
Bill Gates	Detail	Delete
Mark Zuckerberg	Detail	Delete
Sergey Brin	Detail	Delete

Contact Detail Component

Contact detail

Sergey Brin

Projects: Google, Alphabet Inc.

Email: sergey.brin@gmail.com

Website: http://google.com

## Correction

### ▪ Etape 1 : Configuration du serveur : json-server

- **Installation de json-server:** json-server est un node module, pour l'installer taper la commande suivante

```
$ npm install -g json-server@0.17.0
```

- **Configuration du json-server :** Télécharger le fichier **json-server.zip** et l'extraire dans le dossier **AngularProjects**, puis dans votre terminal, accéder au dossier **json-server** et taper la commande suivante :

```
$ json-server --watch db.json -d 2000
```

Cela devrait démarrer un serveur au **port 3000** sur votre machine. Les ressources de ce serveur sont accessibles en saisissant les adresses suivantes dans la barre d'adresse de votre navigateur:

**Resources**  
<http://localhost:3000/contacts>  
<http://localhost:3000/comments>

Remarque : Ces ressources sont obtenues à partir du fichier db.json

©MAD

mohamedanouer.dahdeh@fsb.ucar.tn

## Correction

- **Servir les images:** json-server fournit également un serveur web statique. Toutes les ressources que nous plaçons dans un dossier nommé **public** du dossier **json-server** seront servies par le serveur à l'adresse:

Home  
<http://localhost:3000>

Vous pouvez afficher les images contenues dans le dossier public->images en tapant ce qui suit dans la barre d'adresse de votre navigateur:

<http://localhost:3000/images/<image name>.png>

data (E) > AngularProjects > json-server > public >  

Nom	Modifié
images	03/12/2

©MAD

mohamedanouer.dahdeh@fsb.ucar.tn

## Correction

- **Etape 2 :** Configuration de l'URL du serveur (Json-Server) et HttpClientModule :

1- Créez un nouveau fichier nommé **baseurl.ts** dans le dossier **Shared** et mettez à jour son contenu comment suit :

```
TS baseUrl.ts X
src > app > Shared > TS baseUrl.ts > ...
1 export const BaseURL = 'http://localhost:3000/';
```

2- Importer **BaseUrl** et **HttpClientModule** dans **app.module.ts**

```
TS app.module.ts X
src > app > TS app.module.ts > ...
19 import { EditContactReactiveFormComponent } from './edit-contact-reactive-form/edit-contact-reactive-form-compo
20 import { BaseURL } from './Shared/BaseUrl';
21 import { HttpClientModule } from '@angular/common/http';
22 @NgModule({
23   declarations: [
24     AppComponent,
25     ContactsComponent,
26     AboutComponent,
27     HomeComponent,
28     SignInComponent,
29     NotFoundComponent,
30     ContactDetailComponent,
31     CommentsComponent,
32     EditContactComponent,
33     EditContactReactiveFormComponent
34   ],
35   imports: [
36     BrowserModule,
37     FormsModule,
38     ReactiveFormsModule,
39     AppRoutingModule,
40     HttpClientModule
41   ],
42   providers: [AuthService,
43     ContactService,
44     AuthGuard,
45     AuthService,
46     {provide: 'BaseUrl', useValue: BaseURL}],
47   bootstrap: [AppComponent]
48 })
49 export class AppModule { }
```

©MAD

## Correction

- **Etape 3 :** Mise à jour du fichier **contact.service.ts**

Injection de **HttpClient** dans **contact.service.ts**, pour effectuer des requêtes HTTP en précisant chaque fois la méthode et l'URL.

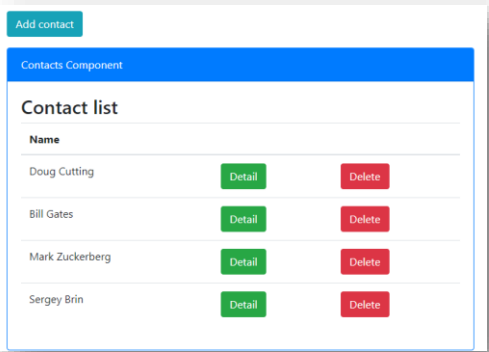
```
TS contact.service.ts X
src > app > services > TS contact.service.ts > ...
1 import { Inject, Injectable } from '@angular/core';
2 import { Contact } from '../Shared/contact';
3 //import { CONTACTS } from '../Shared/contacts';
4 import { HttpClient, HttpHeaders } from '@angular/common/http';
5 import { Observable } from 'rxjs';
6 @Injectable({
7   providedIn: 'root'
8 })
9 export class ContactService {
10   constructor(private httpClient: HttpClient, @Inject('BaseUrl') private baseUrl: string) { }
11
12   getContacts(): Observable<Contact[]> {
13     return this.httpClient.get<Contact[]>(`${this.baseUrl}contacts/`);
14   }
15
16   getContactById(id: number): Observable<Contact> {
17     return this.httpClient.get<Contact>(`${this.baseUrl}contacts/${id}`);
18   }
19
20   deleteContactById(id: number): Observable<any> {
21     return this.httpClient.delete<any>(`${this.baseUrl}contacts/${id}`);
22   }
23
24   addContact(contact: Contact): Observable<Contact> {
25     const httpOptions = {
26       headers: new HttpHeaders({
27         'Content-Type': 'application/json'
28       })
29     };
30     return this.httpClient.post<Contact>(`${this.baseUrl}contacts/`, contact, httpOptions);
31   }
32 }
```

©MAD

# Correction

- Etape 4 : Mise à jour du fichier `contacts.component.ts`

```
18 contacts.component.ts X
src > app > contacts > TS contacts.component.ts > ContactsComponent > ngOnInit
9
10
11 export class ContactsComponent implements OnInit {
12   contacts: Contact[] = [];
13   constructor(private router: Router, private contactService: ContactService) {}
14   ngOnInit(): void {
15     this.contactService.getContacts().subscribe(res => { this.contacts = res; });
16     //this.contacts=this.contactService.getContacts();
17   }
18   onDelete(id: number) {
19     this.contactService.deleteContactById(id).subscribe(res => {
20       //delete the contact from contacts attribute
21       let index = this.contacts.findIndex(contact => contact.id == id);
22       return this.contacts.splice(index, 1);
23     });
24     //this.contactService.deleteContactById(id);
25   }
26   onAbout() {
27     this.router.navigate(['/about']);
28   }
29   onAddContact() {
30     this.router.navigate(['/contacts/edit']);
31   }
32   onAddContactReactiveForm() {
33     this.router.navigate(['/contacts/edit-reactive-form']);
34   }
35 }
```



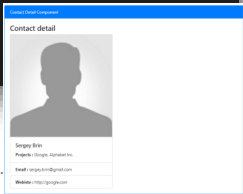
mohamedanouer.dahdeh@fsb.ucar.tn

# Correction

- Etape 5 : Mise à jour du fichier `contact-detail.component.ts` et `contact-detail.component.html`

```
18 contact-detail.component.ts X
src > app > contact-detail > TS contact-detail.component.ts > ...
6
7 selector: 'app-contact-detail',
8 templateUrl: './contact-detail.component.html',
9 styleUrls: ['./contact-detail.component.css']
10
11 export class ContactDetailComponent implements OnInit {
12   idContact: any;
13   contact: Contact;
14
15   constructor(private contactService: ContactService,
16               private route: ActivatedRoute,
17               private router: Router,
18               @Inject('BASEURL') private baseUrl: string) {}
19   ngOnInit(): void {
20     // snapshot method
21     // this.idContact=this.route.snapshot.params.id;
22     // asynchronous method
23     this.route.paramMap.subscribe(result => {
24       this.idContact = result.get('id');
25     });
26     this.contactService.getContactById(this.idContact).subscribe(contact => { this.contact = contact; });
27     //this.contact = this.contactService.getContactById(this.idContact);
28   }
29   onContacts() {
30     this.router.navigate(['/contacts']);
31   }
32 }
```

```
18 contact-detail.component.html X
src > app > contact-detail > contact-detail.component.html > ...
1 <div class="card border-primary mb-3">
2   <div class="card-header bg-primary text-white"> Contact Detail Component</div>
3   <div class="card-body">
4     <div *ngIf="contact; else noContacts">
5       <h3>Contact detail</h3>
6       <div class="card" style="width: 25em;">
7         
8         <div class="card-body">
9           <h4 class="card-title">{{(contact.name)}}</h4>
10          <p class="card-text">{{(contact.projects)}}</p>
11          <ul class="list-group list-group-flush">
12            <li class="list-group-item">{{(contact.email)}}</li>
13            <li class="list-group-item">{{(contact.website)}}</li>
14          </ul>
15        </div>
16      </div>
17    </div>
18    <ng-template #noContacts>
19      <p>Contact not found</p>
20    </ng-template>
21  </div>
22 </div>
23 <button class="btn btn-primary" (click)="onContacts()">
24   Go to Contacts
25 </button>
```



mohamedanouer.

©MAD

# Correction

- Etape 6 : Mise à jour du fichier edit-contact.component.ts

```
11 export class EditContactComponent implements OnInit {
12   constructor(private router: Router, private contactService : ContactService) { }
13   ngOnInit(): void {
14   }
15   onSubmit(form: NgForm){
16     //create a new contact object and initialize it with the values of the form elements
17     let contact : Contact = {
18       //id : -1 ,
19       id : null,
20       name : form.value['name'],
21       email : form.value['email'],
22       website : form.value['website'],
23       projects : [form.value['projects']], //provide a function to extract projects separated by comma
24       featured : false,
25       //image : './assets/images/default-avatar.jpg' //default picture
26       image : 'images/default-avatar.jpg' //default picture
27     };
28     //this.contactService.addContact(contact);
29     this.router.navigate(['/contacts']) // redirect the user to contacts*/
30     this.contactService.addContact(contact).subscribe(
31       contact=>{ // if the contact added with success, redirect the user to contacts
32         // this.router.navigate(['/contacts'])
33       });
34   }
35   onContacts() {
36     this.router.navigate(['/contacts']);
37   }
38 }
```

Edit Contact Component

Add Contact

Name

Sergey Brin

Email

sergey.brin@gmail.com

Website

http://google.com

Projects (separated with comma,"")

Google, Alphabet Inc

Save

©MAD

mohamedanouer.dahdeh@fsb.ucar.tn

# Exercice 2

#9

Entant que utilisateur

authentifié

je veux modifier un contact

©MAD

first-project

Home About Contacts Signin

Add contact

Add contact with reactive form

Contacts Component

Contact list

Name			
Doug Cutting	<div>Detail</div>	<div>Edit</div>	<div>Delete</div>
Bill Gates	<div>Detail</div>	<div>Edit</div>	<div>Delete</div>
Mark Zuckerberg	<div>Detail</div>	<div>Edit</div>	<div>Delete</div>

Go to About

Edit Contact Component

Update Contact

Name

Doug Cutting

Email

doug.cutting@cloudera.com

Website

http://hadoop.apache.org

Projects (separated with comma,"")

Nuchu-Hadoop

Save



## Correction

- **Etape 1 :** Mettre à jour le fichier `contact.service.ts`

```
TS contact.service.ts X
src > app > services > TS contact.service.ts > ...
1 import { Inject, Injectable } from '@angular/core';
2 import { Contact } from '../shared/contact';
3 //import { Contacts } from '../shared/contacts';
4 import { HttpClient, HttpHeaders } from '@angular/common/http';
5 import { Observable } from 'rxjs';
6 @Injectable({
7   providedIn: 'root'
8 })
9 export class ContactService {
10   constructor(private httpClient: HttpClient, @Inject('baseUrl') private baseUrl: string) {}
11
12   getContacts(): Observable<Contact[]> {
13     return this.httpClient.get<Contact[]>(`${this.baseUrl}/contacts/`);
14   }
15
16   getContactById(id: number): Observable<Contact> {
17     return this.httpClient.get<Contact>(`${this.baseUrl}/contacts/${id}`);
18   }
19
20   deleteContactById(id: number): Observable<any> {
21     return this.httpClient.delete<any>(`${this.baseUrl}/contacts/${id}`);
22   }
23
24   addContact(contact: Contact): Observable<Contact> {
25     const httpOptions = {
26       headers: new HttpHeaders({
27         'Content-Type': 'application/json'
28       })
29     };
30     return this.httpClient.post<Contact>(`${this.baseUrl}/contacts/`, contact, httpOptions);
31   }
32 }
```

```
TS contact.service.ts X
src > app > services > TS contact.service.ts > ...
9 export class ContactService {
10   httpOptions = {
11     headers: new HttpHeaders({
12       'Content-Type': 'application/json'
13     })
14   };
15   constructor(private httpClient: HttpClient, @Inject('baseUrl') private baseUrl: string) {}
16   getContacts(): Observable<Contact[]> {
17     return this.httpClient.get<Contact[]>(`${this.baseUrl}/contacts/`);
18   }
19   getContactById(id: number): Observable<Contact> {
20     return this.httpClient.get<Contact>(`${this.baseUrl}/contacts/${id}`);
21   }
22   deleteContactById(id: number): Observable<any> {
23     return this.httpClient.delete<any>(`${this.baseUrl}/contacts/${id}`);
24   }
25   addContact(contact: Contact): Observable<Contact> {
26     return this.httpClient.post<Contact>(`${this.baseUrl}/contacts/`, contact, this.httpOptions);
27   }
28   updateContact(contact: Contact): Observable<Contact> {
29     return this.httpClient.put<Contact>(`${this.baseUrl}/contacts/${contact.id}`, contact, this.httpOptions);
30   }
31 }
```

©MAD

mohamedanouer.dahdeh@fsb.ucar.tn

## Correction

- **Etape 2 :** Mettre à jour le fichier `app-routing.module.ts`:

Nous utiliserons `EditContactComponent` pour ajouter et modifier un contact

- `id = -1` (Ajout d'un nouveau contact)
- `id = idContact` (Mise à jour d'un contact)

```
TS app-routing.module.ts X
src > app > TS app-routing.module.ts > ...
12 const routes: Routes = [
13   { path: '', canActivate: [AuthGuard], component: HomeComponent },
14   { path: 'contacts', canActivate: [AuthGuard], component: ContactsComponent },
15   { path: 'contacts/edit', canActivate: [AuthGuard], component: EditContactComponent },
16   { path: 'contacts/edit-reactive-form', canActivate: [AuthGuard], component: EditContactReactiveFormComponent },
17   { path: 'contacts/:id', canActivate: [AuthGuard], component: ContactDetailComponent },
18   { path: 'about', canActivate: [AuthGuard], component: AboutComponent },
19   { path: 'signin', component: SigninComponent },
20   { path: '**', component: NotFoundComponent }
21 ];
```

```
src > app > TS app-routing.module.ts > ...
12 const routes: Routes = [
13   { path: '', canActivate: [AuthGuard], component: HomeComponent },
14   { path: 'contacts', canActivate: [AuthGuard], component: ContactsComponent },
15   { path: 'contacts/edit/:id', canActivate: [AuthGuard], component: EditContactComponent },
16   { path: 'contacts/edit-reactive-form', canActivate: [AuthGuard], component: EditContactReactiveFormComponent },
17   { path: 'contacts/:id', canActivate: [AuthGuard], component: ContactDetailComponent },
18   { path: 'about', canActivate: [AuthGuard], component: AboutComponent },
19   { path: 'signin', component: SigninComponent },
20   { path: '**', component: NotFoundComponent }
21 ];
```

©MAD

mohamedanouer.dahdeh@fsb.ucar.tn

# Correction

- Etape 3 : Mettre à jour le fichier `contact.component.ts`

TS `contacts.component.ts` X

src > app > contacts > TS `contacts.component.ts` > ...

29

onAddContact() {

30

this.router.navigate(['/contacts/edit/-1']);

31

}

32

onAddContactReactiveForm() {

33

this.router.navigate(['contacts/edit-reactive-form']);

34

}

35

}

first-project

Home About Contacts Signin

Add contact Add contact with reactive form

Contact list

Name

Doug Cutting

Bill Gates

Mark Zuckerberg

Go to About

Edit Contact Component

Add Contact

Name

Email

Website

Projects (separated with comma,")

Save

Go to Contacts

©MAD

# Correction

- Etape 4 : Mettre à jour le fichier `contact.component.html`

`contacts.component.html` X

src > app > contacts > `contacts.component.html` > ...

10

<div \*ngIf="contacts.length > 0; else noContacts">

11

<h3>Contact list</h3>

12

<table class="table table-hover">

13

<thead>

14

<tr>

15

<th scope="col">Name</th>

16

<th scope="col"></th>

17

<th scope="col"></th>

18

<th scope="col"></th>

19

</tr>

20

</thead>

21

<tbody>

22

<tr \*ngFor="let c of contacts">

23

<td>{{c.name}}</td>

24

<td><a class="btn btn-success" routerLink="/contacts/{{c.id}}>Detail</a></td>

25

<td><a class="btn btn-warning" routerLink="/contacts/{{c.id}}>Edit</a></td>

26

<td><a class="btn btn-danger" (click)="onDelete(c.id)">Delete</a></td>

27

</tr>

28

</tbody>

29

</table>

30

</div>

31

<ng-template #noContacts>

32

<p>No Contacts</p>

33

</ng-template>

34

</div>

35

</div>

first-project

Home About Contacts Signin

Add contact Add contact with reactive form

Contacts Component

Contact list

Name

Doug Cutting

Bill Gates

Mark Zuckerberg

Go to About

Detail

Edit

Delete

Detail

Edit

Delete

Detail

Edit

Delete

©MAD

10

## Correction

- Etape 5 : Mettre à jour le fichier `edit-contact.component.ts`

The screenshot displays the `edit-contact.component.ts` file on the left and the rendered UI on the right. The code defines the `EditContactComponent` which implements `OnInit`. It includes a `contact` property, a `constructor` with `Router` and `ContactService`, and an `ngOnInit` method. The `initContact` method subscribes to the contact by ID. The `onSubmit` method handles the form submission, either creating a new contact or updating an existing one, and redirects the user accordingly. The UI on the right shows the 'Update Contact' form with fields for Name, Email, Website, and Projects, a 'Save' button, and a 'Go to Contacts' link.

mohamedanouer.dahdeh@fsb.ucar.tn

## Correction

- Etape 6 : Mettre à jour le fichier `edit-contact.component.html`

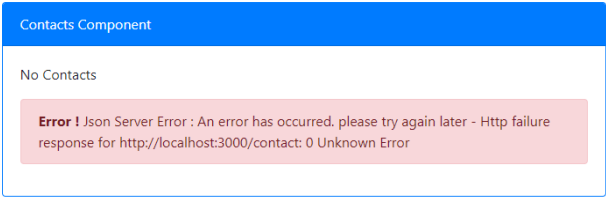
The screenshot displays the `edit-contact.component.html` file on the left and the rendered UI on the right. The HTML template defines the form structure, including input fields for Name, Email, Website, and Projects, and a 'Save' button. The UI on the right shows the rendered form with the same fields and button.

mohamedanouer.dahdeh@fsb.ucar.tn

# Exercice 3

## #6 BNF

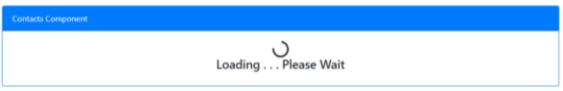
Gérer les erreurs de connexion au serveur



©MAD

## #7 BNF

Ajouter un spinner en attendant la réponse du serveur



mohamedanouer.dahdeh@fsb.ucar.tn

# Correction 3 : Gérer les erreurs

## #6 BNF

- Etape 1 : Créer un service pour gérer les erreurs de connexion au json-server

```
TS processhttpmsg.service.ts X
1 import { HttpResponse } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { throwError } from 'rxjs';
4
5 @Injectable({
6   providedIn: 'root'
7 })
8 export class ProcessHttpmsgService {
9   constructor() {}
10  public handleError(error: HttpResponse | any) {
11    let errMsg: string;
12    // Check if the error is an instance of ErrorEvent (client-side error)
13    if (error.error instanceof ErrorEvent) {
14      errMsg = error.error.message; // Extract the error message from the client-side error
15    } else { // If it's not an instance of ErrorEvent, it's a server-side error
16      // In production, don't display the detailed error message to users
17      // Only display it in development mode for debugging purposes
18      errMsg = `Json Server Error : An error has occurred, please try again later
19      | | | | | - ${error.message || ''}`;
20    }
21    // Return the error message as an Observable using throwError
22    // This allows the error to be caught and handled by the caller
23    return throwError(() => errMsg);
24  }
25 }
```

```
$ ng g s services/process-httpmsg
```

```
TS app.module.ts X
src > app > TS app.module.ts > ...
41 | HttpClientModule
42 | ],
43 | providers: [AboutService,
44 |             ContactService,
45 |             AuthGuard,
46 |             AuthService,
47 |             {provide: 'BaseUrl', useValue: BaseURL},
48 |             ProcessHttpmsgService],
```

©MAD

mohamedanouer.dahdeh@fsb.ucar.tn

## Correction 3 : Gérer les erreurs

#6 BNF

- **Etape 2 :** Capturer les erreurs de connexion avec **Pipe ()** et **CatchError()** lors de l'appel d'une requête HTTP

```

TS contact.service.ts X
src > app > services > TS contact.service.ts > ...
1 import { Inject, Injectable } from '@angular/core';
2 import { Contact } from '../Shared/contact';
3 //import { CONTACTS } from '../Shared/contacts';
4 import { HttpClient, HttpHeaders } from '@angular/common/http';
5 import { Observable } from 'rxjs';
6 import { ProcessHttpmsgService } from '../process-httpmsg.service';
7 import { catchError } from 'rxjs/operators';
8 @Injectable({
9   providedIn: 'root'
10 })
11 export class ContactService {
12   httpOptions = {
13     headers: new HttpHeaders({
14       'Content-Type': 'application/json'
15     })
16   };
17   constructor(private httpClient: HttpClient, @Inject('BaseUrl') private baseUrl,
18     private processHttpMsgService : ProcessHttpmsgService ) { }
19   getContacts(): Observable<Contact[]> {
20     return this.httpClient.get<Contact[]>(this.baseUrl + "contacts/").
21       pipe(catchError(this.processHttpMsgService.handleError));
22   }
23   getContactById(id: number): Observable<Contact> {

```

©MAD

er.dahdeh@fsb.ucar.tn

## Correction 3 : Gérer les erreurs

#6 BNF

- **Etape 3 :** Gérer les erreurs dans subscribe ()

```

TS contacts.component.ts U X
6 @Component({
7   selector: 'app-contacts',
8   templateUrl: './contacts.component.html',
9   styleUrls: ['./contacts.component.css']
10 })
11 export class ContactsComponent implements OnInit {
12   contacts: Contact[];
13   errMsg: string;
14   public constructor(private router: Router,
15     private contactService: ContactService) { }
16   ngOnInit(): void {
17     this.contactService
18       .getContacts()
19       .subscribe({
20         next: (contacts) => { this.contacts = contacts; },
21         error: (errMsg) => {
22           this.contacts = [];
23           this.errMsg = <any>errMsg;
24         },
25         complete: () => { console.log("Complete"); }
26       });
27   }

```

©MA

```

contacts.component.html X
src > app > contacts > contacts.component.html > ...
28 </tbody>
29 </table>
30 </div>
31 <ng-template #noContacts>
32   <p>No Contacts</p>
33   <div *ngIf="errMsg">
34     <h2>Error</h2>
35     <h4>{{errMsg}}</h4>
36   </div>
37 </ng-template>
38 </div>
39 </div>
40
41 <button class="btn btn-primary" (click)="onAbout()">
42   Go to About
43 </button>

```

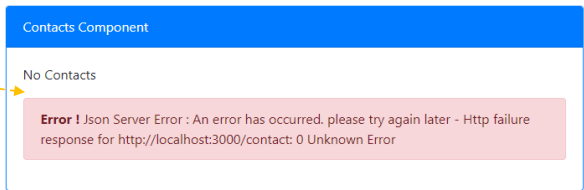
mohamedanouer.dahdeh@fsb.ucar.tn

# Correction 3 : Gérer les erreurs

#6 BNF

- Etape 4 : Afficher les erreurs dans la vue du composant

```
contacts.component.html X
src > app > contacts > contacts.component.html > ...
27 |         </tr>
28 |     </tbody>
29 | </table>
30 | </div>
31 | <ng-template #noContacts>
32 |     <p>No Contacts</p>
33 |     <div *ngIf="errMess" class="alert alert-danger">
34 |         <strong>Error !</strong> {{errMess}}.
35 |     </div>
36 | </ng-template>
37 | </div>
38 | </div>
39 |
40 | <button class="btn btn-primary" (click)="onAbout()">
41 |     Go to About
42 | </button>
```



©MAD

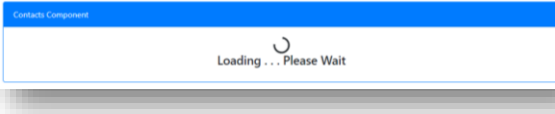
mohamedanouer.dahdeh@fsb.ucar.tn

# Correction 3 : Gérer l'attente du serveur

#7 BNF

```
TS contacts.component.ts X
11 export class ContactsComponent implements OnInit {
12     contacts: Contact[];
13     errMess: string;
14     isWaiting: boolean = true;
15     public constructor(private router: Router,
16         private contactService: ContactService) { }
17     ngOnInit(): void {
18         this.contactService
19             .getContacts()
20             .subscribe({
21                 next: (contacts) => {
22                     this.contacts = contacts;
23                     this.isWaiting = false;
24                 },
25                 error: (errmess) => {
26                     this.contacts = [];
27                     this.errMess = <any>errmess;
28                     this.isWaiting = false;
29                 },
30                 complete: () => { console.log("Complete"); }
31             });
32 }
```

```
contacts.component.html X
29 |         </table>
30 |     </div>
31 |     <div [hidden]="!isWaiting" class="text-center">
32 |         <div class="spinner-border" role="status"></div>
33 |         <h4>Loading... Please Wait</h4>
34 |     </div>
35 |     <ng-template #noContacts>
36 |         <!-- <p>No Contacts</p> -->
37 |         <p [hidden]="isWaiting">No Contacts</p>
38 |         <div *ngIf="errMess" class="alert alert-danger">
39 |             <strong>Error ! </strong> {{errMess}}.
40 |         </div>
41 |     </ng-template>
42 | </div>
43 | </div>
```



©MAD

## En savoir plus !

- A la place de **json-server**, vous pouvez utiliser le service **FireBase** de google :
  - Authentification
  - Une base de données
  - Stockage de Fichiers
- Ou implementer votre API REST à l'aide d'un framework backend comme Spring Boot
- Pour finir vous allez vouloir faire un build de votre application pour préparer vos livrables et c'est la commande **ng build** qui va vous y aider

```
$ ng build --prod
```

©MAD

mohamedanouer.dahdeh@fsb.ucar.tn

**03**

## Atelier 5

