08

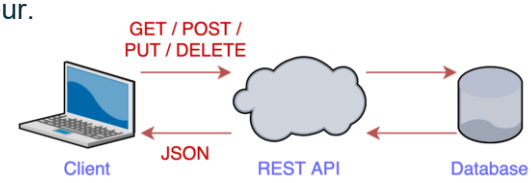
Communication Client – serveur

Chapitre 8 Communication Client-Serveur

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.

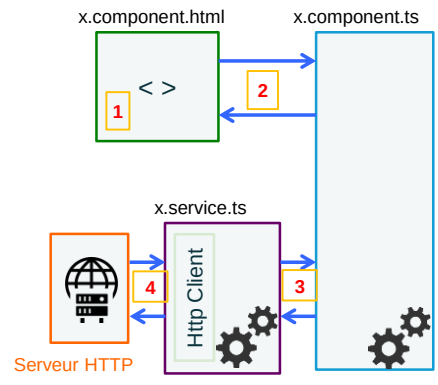


Angular & les serveurs

- 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>`

Principe

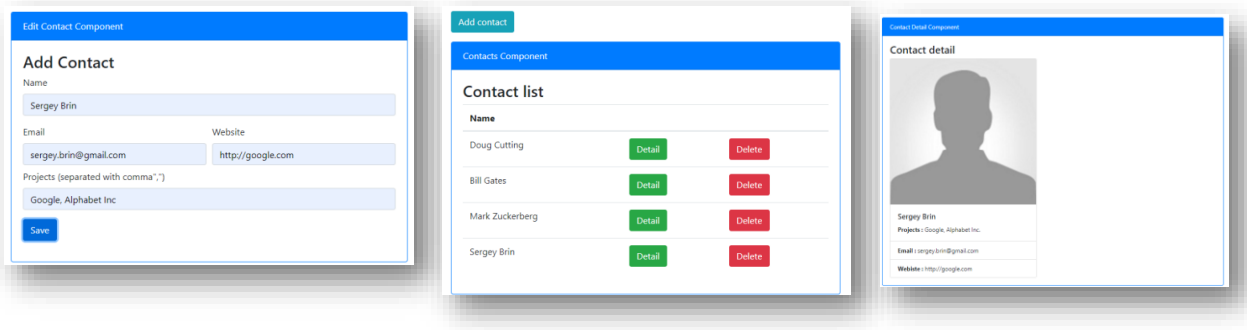
1. Les données sont saisies (ou affichées) dans le Template .component.html
2. La classe .component.ts peut récupérer des données dont la source est le Template .component.html pour les passer au service ou récupérer des données dont la source est le service pour les passer au template .component.html.
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.



02 Appeler un Backend

Exercice 1 : Communication client-serveur

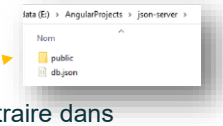
- Récupérer la liste des contacts à partir d'une API REST **(json-server)**
- Sauvegarder tous les changements dans une API REST
  - Ajout et Suppression



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:

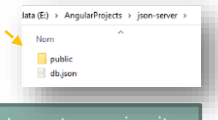
```
Resources
http://localhost:3000/contacts
http://localhost:3000/comments
```

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

Correction

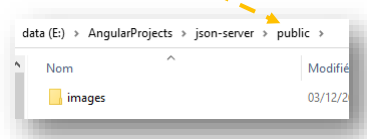
- **Servir les images:** json-server fournit également un serveur web statique. Toutes les ressources que nous plaçons dans un dossier nommé **public** du dossier **json-server** seront servies par le serveur à l'adresse:

```
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



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 ×                                                    1
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 ×
src > app > TS app.module.ts > ...
import { EditContactReactiveFormComponent } from './edit-contact-reactive-form/edit-contact-reactive-form.compc
import { BaseURL } from './Shared/baseUrl';
import { HttpClientModule } from '@angular/common/http';
@NgModule({
  declarations: [
    AppComponent,
    ContactsComponent,
    AboutComponent,
    HomeComponent,
    SignInComponent,
    NotfoundComponent,
    ContactDetailComponent,
    CommentsComponent,
    EditContactComponent,
    EditContactReactiveFormComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    ReactiveFormsModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [AboutService,
              ContactService,
              AuthGuard ,
              AuthService,
              {provide: 'BaseURL', useValue: BaseURL}],

  bootstrap: [AppComponent]
})
export class AppModule { }                                                    2
```


Correction

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

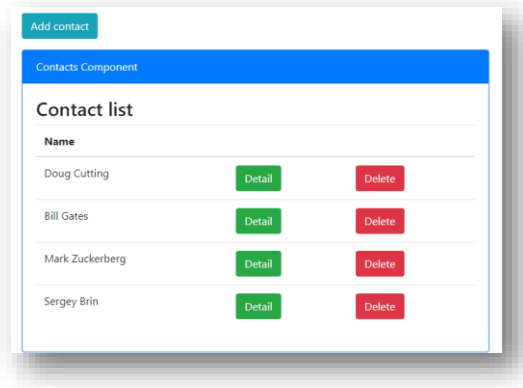
Injection de `HttpClient` dans `contcat.service.ts`, pour effectuer des requêtes HTTP en précisant chaque fois **la méthode et l'URL.**

```
TS contact.service.ts ×
src > app > services > TS contact.service.ts > ...
import { Inject, Injectable } from '@angular/core';
import { Contact } from '../Shared/contact';
//import { CONTCATS } from '../Shared/contacts';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs';
@Injectable({
  providedIn: 'root'
})
export class ContactService {
  constructor(private httpClient: HttpClient, @Inject('BaseURL') private baseURL) { }

  getContacts(): Observable<Contact[]> {
    return this.httpClient.get<Contact[]>(this.baseURL + "contacts/")
  }

  getContactById(id: number): Observable<Contact> {
    return this.httpClient.get<Contact>(this.baseURL + "contacts/" + id)
  }

  deleteContactById(id: number): Observable<any> {
    return   this.httpClient.delete <any> (this.baseURL+"contacts/"+id);
  }

  addContact(contact: Contact) : Observable <Contact> {
    const httpOptions = {
      headers: new HttpHeaders({
        'Content-Type':  'application/json'
      })
    };
    return   this.httpClient.post<Contact> (this.baseURL+"contacts/",contact,httpOptions);
  }
}
```

Correction

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

```ts
// contacts.component.ts
  styleUrls: ['./contacts.component.css']
})
export class ContactsComponent implements OnInit {
  contacts: Contact[]=[];
  constructor(private router: Router, private contactService: ContactService) { }
  ngOnInit(): void {
    this.contactService.getContacts().subscribe(res => { this.contacts = res;});
    //this.contacts=this.contactService.getContacts();
  }
  onDelete(id: number) {
    this.contactService.deleteContactById(id).subscribe(res => {
      //delete the contact from contacts attribute
      let index = this.contacts.findIndex(contact => contact.id == id);
      return this.contacts.splice(index, 1);
    });
    //this.contactService.deleteContactById(id);
  }
  onAbout() {
    this.router.navigate(['/about']);
  }
  onAddContact() {
    this.router.navigate(['/contacts/edit']);
  }
  onAddContactReactiveForm() {
    this.router.navigate(['contacts/edit-reactive-form']);
  }
}
```



Correction

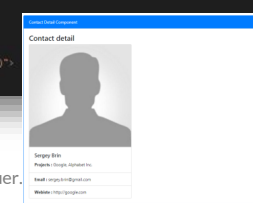
- **Etape 5** : Mise à jour du fichier `contact-detail.component.ts` et `contact-detail.component.html`

```ts
// contact-detail.component.ts
@Component({
  selector: 'app-contact-detail',
  templateUrl: './contact-detail.component.html',
  styleUrls: ['./contact-detail.component.css']
})
export class ContactDetailComponent implements OnInit {
  idContact: any
  contact: Contact;

  constructor(private contactService: ContactService,
              private route: ActivatedRoute,
              private router: Router,
              @Inject('BaseURL') private baseURL) { }
  ngOnInit(): void {
    /** snapshot method */
    // this.idContact=this.route.snapshot.params.id;
    /** asynchronous method */
    this.route.paramMap.subscribe(result => {
      this.idContact = result.get('id');
    })
    this.contactService.getContactById(this.idContact).subscribe(contact => { this.contact = contact; });
    // this.contact = this.contactService.getContactById(this.idContact);
  }
  onContacts() {
    this.router.navigate(['/contacts']);
  }
}
```

1

```html
<!-- contact-detail.component.html -->
<div class="card border-primary mb-3">
    <div class="card-header bg-primary text-white "> Contact Detail Component</div>
    <div class="card-body">
        <div *ngIf="contact ; else noContacts">
            <h3>Contact detail</h3>
            <div class="card" style="width: 25em;">
                <img src="{{baseURL + contact.image}}" class="card-img-top" alt="...">
                <div class="card-body">
                    <h5 class="card-title">{{contact.name}}</h5>
                    <p class="card-text"><strong>Projects :</strong> {{contact.projects}}.</p>
                </div>
                <ul class="list-group list-group-flush">
                    <li class="list-group-item"><strong>Email : </strong>{{contact.email}}</li>
                    <li class="list-group-item"><strong>Webiste : </strong>{{contact.website}}</li>
                </ul>
            </div>
        </div>
        <ng-template #noContacts>
            <p>Contact not found</p>
        </ng-template>
    </div>
</div>
<button class="btn btn-primary" (click)="onContacts()">
    Go to Contacts
</button>
```

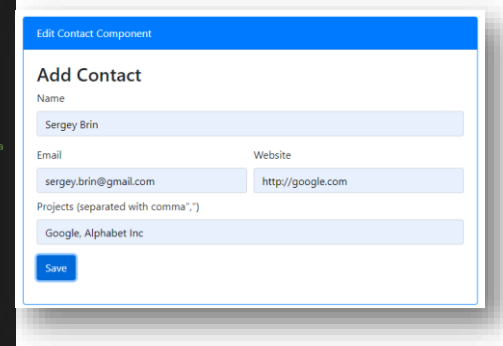
2



Correction

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

```ts
export class EditContactComponent implements OnInit {
  constructor(private router: Router, private contactService : ContactService) { }
  ngOnInit(): void {
  }
  onSubmit(form: NgForm){
    //create a new contact objetc and initialize it with the values of the form elements
    let contact : Contact ={
      //id : -1 ,
      id : null,
      name : form.value['name'],
      email : form.value ['email'],
      website : form.value ['website'],
      projects : [form.value['projects']], //provide a function to extract projects separated by comma
      featured : false,
      //image : './assets/images/default-avatar.jpg' //default picture
      image : 'images/default-avatar.jpg' //default picture
    };
    /*this.contactService.addContact(contact);
    this.router.navigate(['/contacts']) // redirect the user to contacts*/
    this.contactService.addContact(contact).subscribe(
      contact=>{ // if the contact added with success, redirect the user to contacts
                  this.router.navigate(['/contacts'])
                });
  }
  onContacts() {
    this.router.navigate(['/contacts']);
  }
}
```

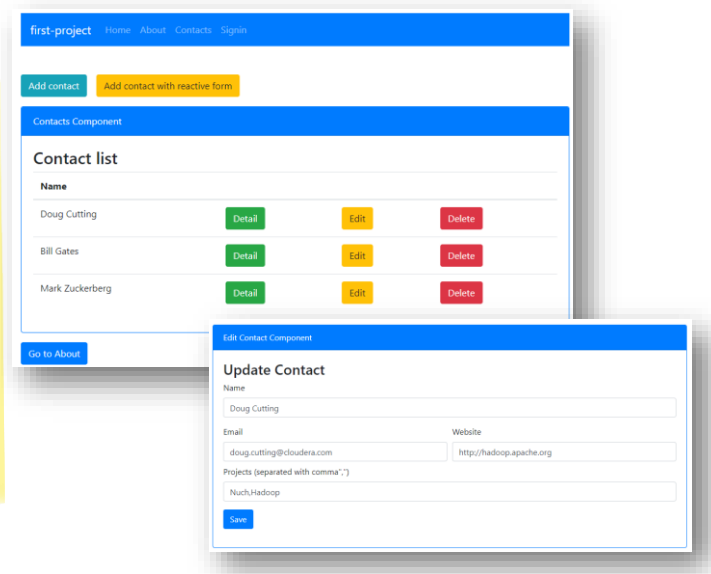


Exercice 2

#9

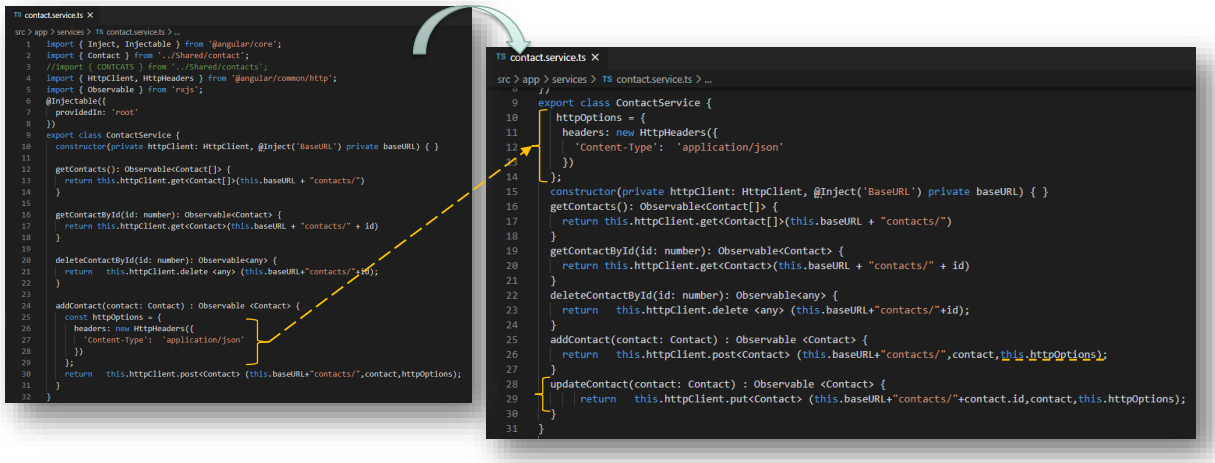
Entant que utilisateur authentifié

je veux modifier un contact



Correction

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



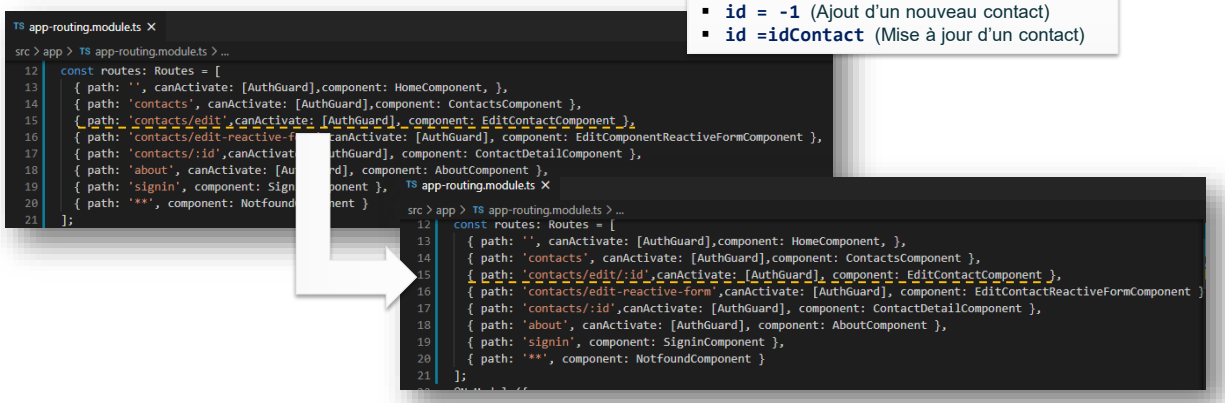
```ts
import { Inject, Injectable } from '@angular/core';
import { Contact } from '../Shared/contact';
//import { CONTACTS } from '../Shared/contacts';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs';
@Injectable({
  providedIn: 'root'
})
export class ContactService {
  constructor(private httpClient: HttpClient, @Inject('BaseURL') private baseURL) { }

  getContacts(): Observable<Contact[]> {
    return this.httpClient.get<Contact[]>(this.baseURL + "contacts/")
  }

  getContactById(id: number): Observable<Contact> {
    return this.httpClient.get<Contact>(this.baseURL + "contacts/" + id)
  }

  deleteContactById(id: number): Observable<any> {
    return   this.httpClient.delete <any> (this.baseURL+"contacts/"+id);
  }

  addContact(contact: Contact) : Observable <Contact> {
    const httpOptions = {
      headers: new HttpHeaders({
        'Content-Type':  'application/json'
      })
    };
    return   this.httpClient.post<Contact> (this.baseURL+"contacts/",contact,httpOptions);
  }
}
```

```ts
export class ContactService {
  httpOptions = {
    headers: new HttpHeaders({
      'Content-Type':  'application/json'
    })
  };
  constructor(private httpClient: HttpClient, @Inject('BaseURL') private baseURL) { }
  getContacts(): Observable<Contact[]> {
    return this.httpClient.get<Contact[]>(this.baseURL + "contacts/")
  }
  getContactById(id: number): Observable<Contact> {
    return this.httpClient.get<Contact>(this.baseURL + "contacts/" + id)
  }
  deleteContactById(id: number): Observable<any> {
    return   this.httpClient.delete <any> (this.baseURL+"contacts/"+id);
  }
  addContact(contact: Contact) : Observable <Contact> {
    return   this.httpClient.post<Contact> (this.baseURL+"contacts/",contact,this.httpOptions);
  }
  updateContact(contact: Contact) : Observable <Contact> {
      return   this.httpClient.put<Contact> (this.baseURL+"contacts/"+contact.id,contact,this.httpOptions);
  }
}
```


Correction

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

Nous utiliserons **`EditContcatComponent`** pour **ajouter et modifier** un contact
- **`id = -1`** (Ajout d'un nouveau contact)
- **`id =idContact`** (Mise à jour d'un contact)

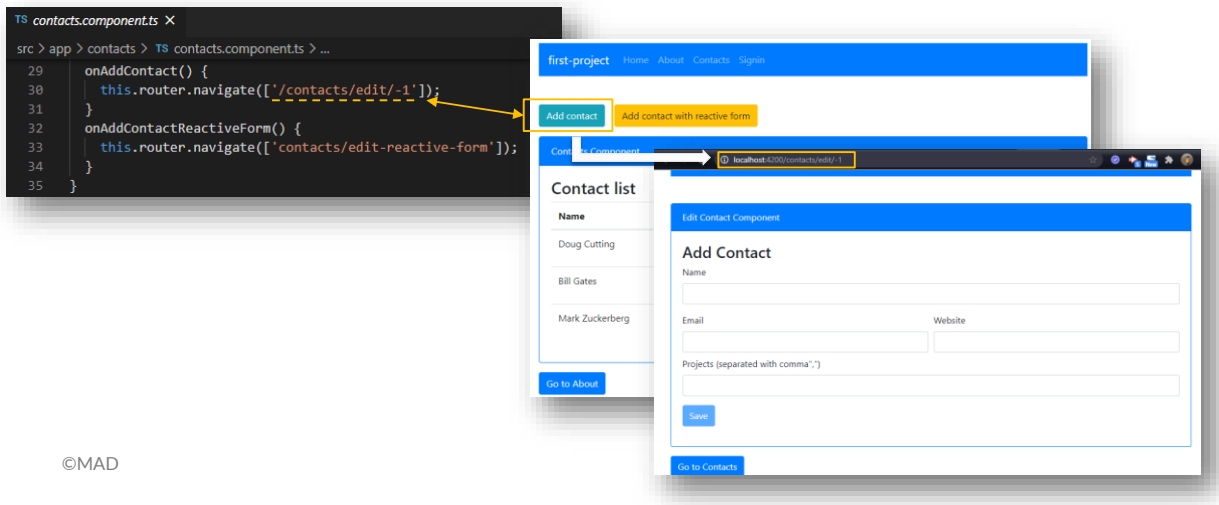


```ts
const routes: Routes = [
  { path: '', canActivate: [AuthGuard],component: HomeComponent, },
  { path: 'contacts', canActivate: [AuthGuard],component: ContactsComponent },
  { path: 'contacts/edit',canActivate: [AuthGuard], component: EditContactComponent },
  { path: 'contacts/edit-reactive-f
  { path: 'contacts/:id',canActivat
  { path: 'about', canActivate: [Au
  { path: 'signin', component: Sign
  { path: '**', component: Notfound
];
```

```ts
const routes: Routes = [
  { path: '', canActivate: [AuthGuard],component: HomeComponent, },
  { path: 'contacts', canActivate: [AuthGuard],component: ContactsComponent },
  { path: 'contacts/edit/:id',canActivate: [AuthGuard], component: EditContactComponent },
  { path: 'contacts/edit-reactive-form',canActivate: [AuthGuard], component: EditContactReactiveFormComponent },
  { path: 'contacts/:id',canActivate: [AuthGuard], component: ContactDetailComponent },
  { path: 'about', canActivate: [AuthGuard], component: AboutComponent },
  { path: 'signin', component: SigninComponent },
  { path: '**', component: NotfoundComponent }
];
```

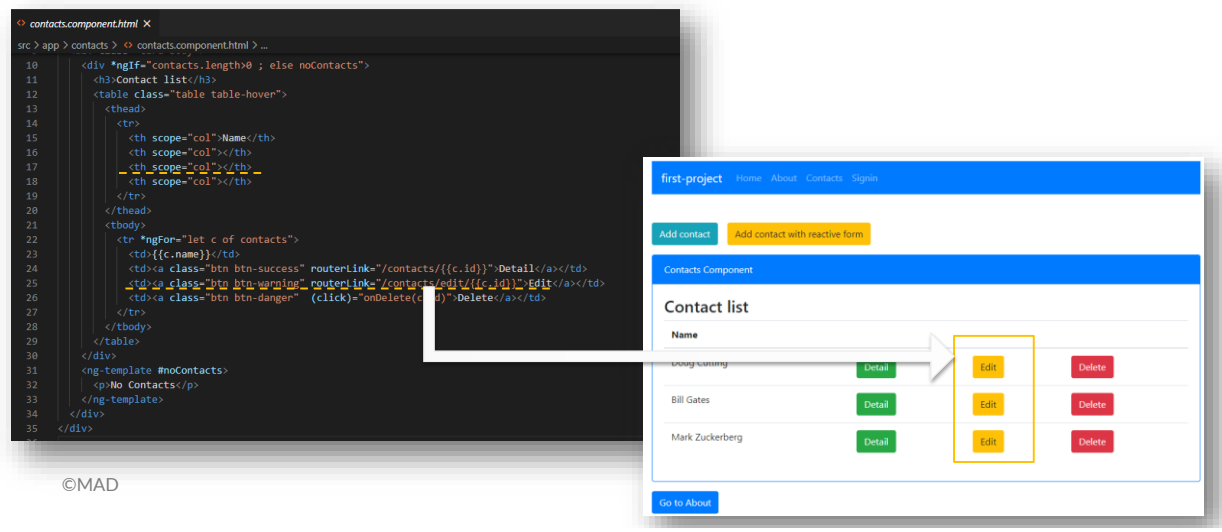
Correction

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



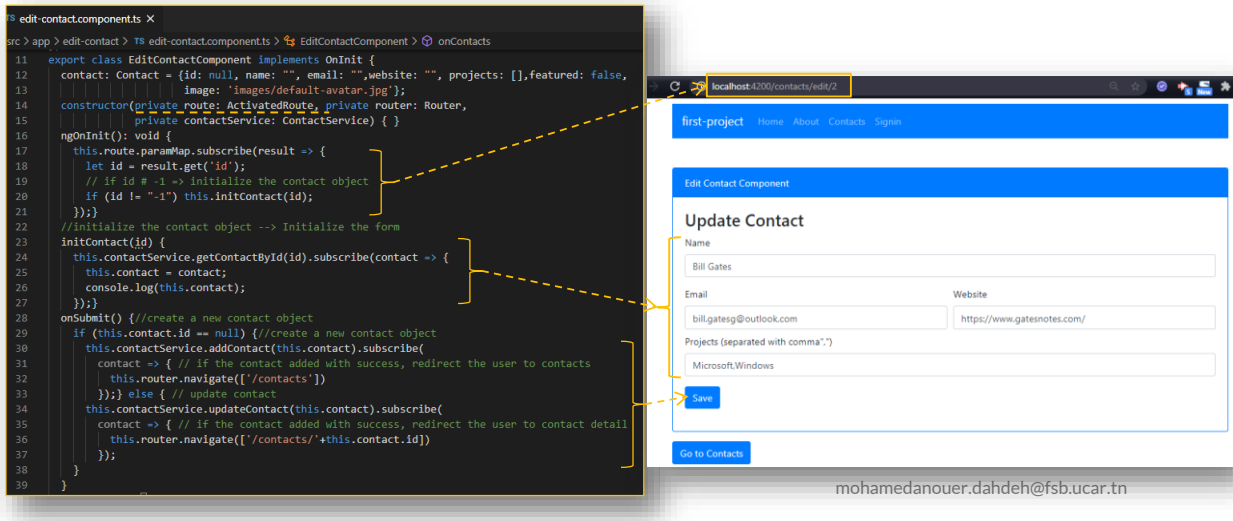
Correction

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



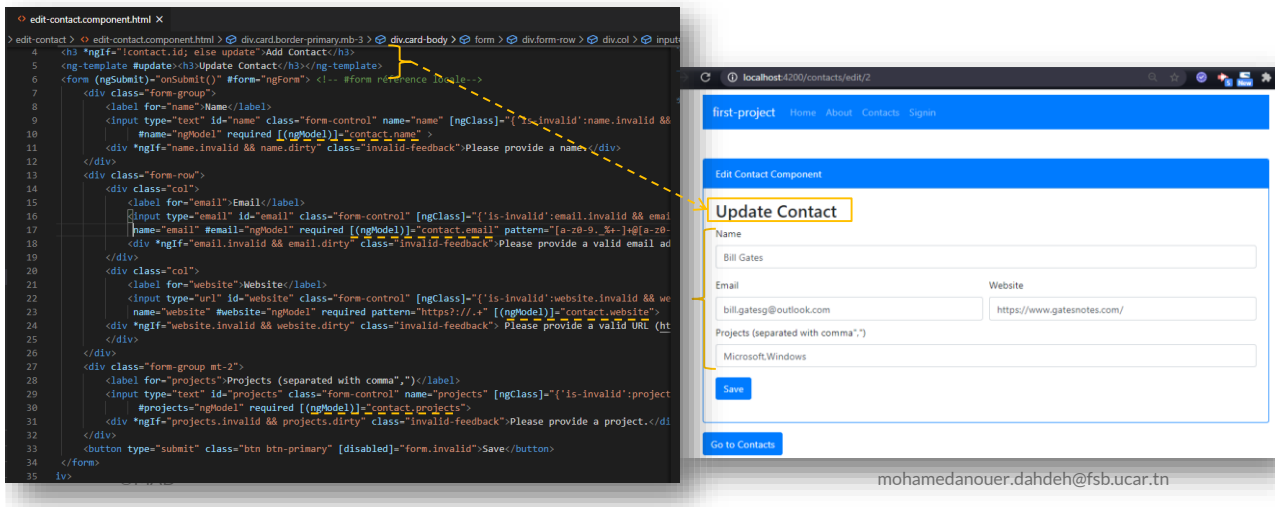
Correction

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



Correction

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



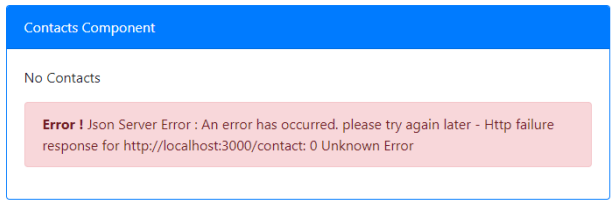
Exercice 3

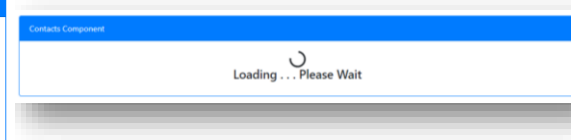
#6 BNF

Gérer **les erreurs de connexion** au serveur

#7 BNF

Ajouter **un spinner** en attendant la réponse du serveur





Correction 3 : Gérer les erreurs #6 BNF

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

processhttpmsg.service.ts U

```
import { HttpErrorResponse } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { throwError } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class ProcesshttpmsgService {
  constructor() { }
  public handleError(error: HttpErrorResponse | any) {
    let errMsg: string;
    // Check if the error is an instance of ErrorEvent (client-side error)
    if (error.error instanceof ErrorEvent) {
      errMsg = error.error.message; // Extract the error message from the client-side error
    } else { // If it's not an instance of ErrorEvent, it's a server-side error
      // In production, don't display the detailed error message to users
      // Only display it in development mode for debugging purposes
      errMsg = `Json Server Error : An error has occurred. please try again later
                - ${error.message || ''} `;
    }
    // Return the error message as an Observable using throwError
    // This allows the error to be caught and handled by the caller
   return throwError(()=>errMsg)
  }
}
```

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

app.module.ts

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

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

contact.service.ts

src > app > services > contact.service.ts > ...

```ts
import { Inject, Injectable } from '@angular/core';
import { Contact } from '../Shared/contact';
//import { CONTCATS } from '../Shared/contacts';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs';
import { ProcessHttpmsgService } from './process-httpmsg.service';
import {catchError } from 'rxjs/operators';
@Injectable({
  providedIn: 'root'
})
export class ContactService {
   httpOptions = {
    headers: new HttpHeaders({
      'Content-Type':  'application/json'
    })
  };
  constructor(private httpClient: HttpClient, @Inject('BaseURL') private baseURL,
              private processHTTPMsgService : ProcessHttpmsgService ) { }
  getContacts(): Observable<Contact[]> {
    return this.httpClient.get<Contact[]>(this.baseURL + "contacts/").
    pipe(catchError(this.processHTTPMsgService.handleError));
  }
  getContactById(id: number): Observable<Contact> {
```


Correction 3 : Gérer les erreurs

#6 BNF

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

contacts.component.ts U

```ts
@Component({
  selector: 'app-contacts',
  templateUrl: './contacts.component.html',
  styleUrls: ['./contacts.component.css']
})
export class ContactsComponent implements OnInit {
  contacts: Contact[];
  errMess: string;
  public constructor(private router: Router,
                     private contactService: ContactService) { }
  ngOnInit(): void {
    this.contactService
      .getContacts()
      .subscribe({
        next: (contacts) => { this.contacts = contacts; },
        error: (errmess) => {
          this.contacts = [];
          this.errMess = <any>errmess;
        },
        complete: () => { console.log("Complete"); }
      });
  }
```

contacts.component.html

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

```html
            </tr>
          </tbody>
        </table>
      </div>
      <ng-template #noContacts>
        <p>No Contacts</p>
        <div *ngIf="errMess">
          <h2>Error</h2>
          <h4>{{errMess}}</h4>
        </div>
      </ng-template>
    </div>
  </div>

  <button class="btn btn-primary" (click)="onAbout()">
    Go to About
  </button>
```

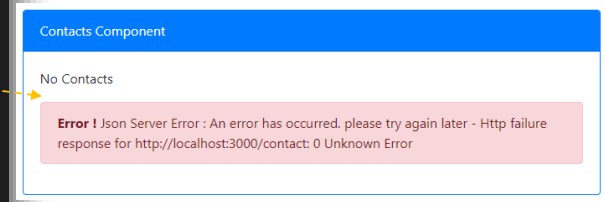
Correction 3 : Gérer les erreurs

#6 BNF

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

contacts.component.html

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



Correction 3 : Gérer l'attente du serveur

#7 BNF

contacts.component.ts U

```
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    }
```

1

contacts.component.html U

```
29          </table>
30        </div>
31        <div [hidden]="!isWaiting" class="text-center">
32          <div class="spinner-border" role="status"></div>
33          <h4>Looding... 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>
```

2

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
```


mohamedanouer.dahdeh@fsb.ucar.tn

03

Atelier 5