

Cours Angular : Le Module HTTP

1. Introduction

Le **module HttpClient** d'Angular permet de réaliser des requêtes HTTP pour communiquer avec un backend.

Objectifs : - Comprendre le fonctionnement du module HTTP - Faire des requêtes GET, POST, PUT, DELETE - Gérer les en-têtes et paramètres - Utiliser les Observables pour l'asynchrone - Gérer les erreurs et intercepteurs

2. Configuration du module HTTP

Dans `app.module.ts` :

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

@NgModule({
  imports: [HttpClientModule]
})
export class AppModule {}
```

3. Injection de HttpClient

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';

@Injectable({ providedIn: 'root' })
export class ApiService {
  constructor(private http: HttpClient) {}
}
```

4. Requêtes HTTP de base

4.1 GET

```
getUsers() {
  return this.http.get<User[]>('/api/users');
}
```

4.2 POST

```
addUser(user: User) {  
  return this.http.post('/api/users', user);  
}
```

4.3 PUT

```
updateUser(id: number, user: User) {  
  return this.http.put(`/api/users/${id}`, user);  
}
```

4.4 DELETE

```
deleteUser(id: number) {  
  return this.http.delete(`/api/users/${id}`);  
}
```

5. Requêtes avec paramètres et en-têtes

5.1 Paramètres URL

```
this.http.get('/api/users', { params: { role: 'admin' } });
```

5.2 En-têtes HTTP

```
import { HttpHeaders } from '@angular/common/http';  
  
const headers = new HttpHeaders().set('Authorization', 'Bearer token');  
this.http.get('/api/users', { headers });
```

6. Gestion des erreurs

```
import { catchError } from 'rxjs/operators';  
import { throwError } from 'rxjs';  
  
this.http.get('/api/users').pipe(  
  catchError(error => {  
    console.error('Erreur HTTP', error);  
    return throwError(error);  
  })
```

```
    })  
  );
```

7. Observables et HTTP

- Les requêtes HTTP retournent **des Observables**
- On peut utiliser `async pipe` dans le template

```
users$ = this.apiService.getUsers();
```

```
<ul>  
  <li *ngFor="let user of users$ | async">{{ user.name }}</li>  
</ul>
```

8. Intercepteurs HTTP

8.1 Principe

- Modifier toutes les requêtes / réponses HTTP globalement
- Ajouter des tokens, logs, gérer erreurs

8.2 Exemple simple

```
import { Injectable } from '@angular/core';  
import { HttpInterceptor, HttpRequest, HttpHandler } from '@angular/common/  
http';  
  
@Injectable()  
export class AuthInterceptor implements HttpInterceptor {  
  intercept(req: HttpRequest<any>, next: HttpHandler) {  
    const cloned = req.clone({ headers: req.headers.set('Authorization',  
'Bearer token') });  
    return next.handle(cloned);  
  }  
}
```

Déclaration dans `app.module.ts` :

```
providers: [  
  { provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true }  
]
```

9. Bonnes pratiques

- Toujours utiliser `HttpClient` (plus sûr que `Http`)
 - Gérer les erreurs avec `catchError`
 - Utiliser des services pour centraliser les requêtes
 - Utiliser `async` pipe pour gérer les Observables dans le template
 - Sécuriser les requêtes avec des intercepteurs
-

10. Cas pratique complet

Objectif

Service HTTP complet pour gérer les utilisateurs :

```
@Injectable({ providedIn: 'root' })
export class UserService {
  constructor(private http: HttpClient) {}

  getUsers() {
    return this.http.get<User[]>('/api/
users').pipe(catchError(this.handleError));
  }

  addUser(user: User) {
    return this.http.post('/api/users',
user).pipe(catchError(this.handleError));
  }

  private handleError(error: any) {
    console.error('Erreur API', error);
    return throwError(error);
  }
}
```

Template :

```
<ul>
  <li *ngFor="let user of users$ | async">{{ user.name }}</li>
</ul>
```

11. Résumé

- Angular `HttpClient` = requêtes HTTP simplifiées et asynchrones
- Utilise des **Observables** pour gérer les données
- Intercepteurs pour sécuriser et modifier globalement les requêtes

- Bonnes pratiques : services centralisés, gestion erreurs, `async` pipe