Angular CRUD 1: IN-MEMORY-WEB-API installation and configuration

Page • 1 backlink • Tag

INSTALLATION

Create a new Angular application.

TypeScript

```typescript
ng new Simple-HR
```

Install the in-memory-web-api:

Plain Text

```
npm i angular-in-memory-web-api@0.8.0 -D
```

The -D flag ensures that this is only a dependency in the development environment.

Configuration

- we create class

TypeScript

```typescript
ng generate class classes/List-of-users
```

TypeScript

```typescript
export class Villain {
    constructor(public id = 0, public name = '', public episode = '' ) { }
}
```

- list-of-users is the blue print for the data available about a users; a primary key (id) along with their name and others.

Services

TypeScript

```typescript
ng g s services/company-in-mem-data
```

TypeScript

```typescript
import { Injectable } from '@angular/core';
import { InMemoryDbService } from 'angular-in-memory-web-api';
import { Company } from '../classes/company';

@Injectable({
    providedIn: 'root'
})
export class CompanyInMemDataService implements InMemoryDbService {
    createDb() {
        let companies: Company[] = [
            { id: 1, name: 'Tech Innovators', industry: 'Technology', employees: 100, address: '123 Tech Road' },
            { id: 2, name: 'Green Solutions', industry: 'Environmental', employees: 50, address: '456 Green Street' },
            { id: 3, name: 'HealthCare Inc.', industry: 'Healthcare', employees: 200, address: '789 Health Ave' },
            { id: 4, name: 'Finance Experts', industry: 'Finance', employees: 75, address: '101 Finance 
Blvd' },
        ];
        return { companies };
    }
}
```

- The service CompanyMemDataService implements inMemoryDbService interface. this example meets the minimum requiremnts for the interface which is the createDb() method.
- This method creates a 'database' containing the initial values of the company and the name , industry , employees and address . At the time of writing the maintainers of in memory web api assume that every collection has a primary key called id

TypeScript

```typescript
ng g s Services/company
```

TypeScript

```typescript
import { Company } from '../classes/company';
import { Observable } from 'rxjs';

export abstract class CompanyService {
    companiesUrl = 'api/companies';  // Updated URL

    abstract getCompanies(): Observable<Company[]>;
    abstract getCompany(id: number): Observable<Company>;
    abstract addCompany(id: number, name: string, industry: string, employees: number, address: string): 
Observable<Company>;
    abstract deleteCompany(company: Company | number): Observable<Company>;
    abstract searchCompanies(term: string): Observable<Company[]>;
    abstract updateCompany(id: number, name: string, industry: string, employees: number, address: string): 
Observable<Company>;
}
```

- Here is an abstract class that defines which operations we want to be able to perform on our 'database' such as listing all the company(getCompany) or adding a new one (addCompany)

http-client-company.service.ts

TypeScript

```
ng g s services/http-client-company
```

TypeScript

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

import { Observable, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';
import { Company } from '../classes/company';  // Updated import
import { CompanyService } from './company.service';  // Updated import

const cudOptions = { headers: new HttpHeaders({ 'Content-Type': 'application/json' }) };

@Injectable({
    providedIn: 'root'
})
export class HttpClientCompanyService extends CompanyService {
    companiesUrl = 'api/companies';  // Updated URL
```

```typescript
    constructor(private http: HttpClient) {
        super();
    }

    getCompanies(): Observable<Company[]> {
        return this.http.get<Company[]>(this.companiesUrl).pipe(
            catchError(this.handleError)
        );
    }

    getCompany(id: number): Observable<Company> {
        const url = `${this.companiesUrl}/${id}`;
        return this.http.get<Company>(url).pipe(
            catchError(this.handleError)
        );
    }

    addCompany(id: number, name: string, industry: string, employees: number, address: string):
Observable<Company> {
        const company = { id, name, industry, employees, address };

        return this.http.post<Company>(this.companiesUrl, company, cudOptions).pipe(
            catchError(this.handleError)
        );
    }

    deleteCompany(company: number | Company): Observable<Company> {
        const id = typeof company === 'number' ? company : company.id;
        const url = `${this.companiesUrl}/${id}`;

        return this.http.delete<Company>(url, cudOptions).pipe(
            catchError(this.handleError)
        );
    }

    searchCompanies(term: string): Observable<Company[]> {
        term = term.trim();
        const options = term ? { params: new HttpParams().set('name', term) } : {};

        return this.http.get<Company[]>(this.companiesUrl, options).pipe(
            catchError(this.handleError)
        );
    }

    updateCompany(id: number, name: string, industry: string, employees: number, address: string):
Observable<Company> {
        const company = { id, name, industry, employees, address };

        return this.http.put<Company>(this.companiesUrl, company, cudOptions).pipe(
            catchError(this.handleError)
        );
    }

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

- this service implements the methods of the companyService abstract class. each operation is an http request that will be handled by the in-memory-web-api

- all the crud operations implemented so at this point you can go off and build you own user interface on top of this service.

Changes To App.Modules.ts

TypeScript

```typescript
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { ListCompaniesComponent } from './components/list-company/list-company.component';
import { CreateCompanyComponent } from './components/create-company/create-company.component';
import { UpdateCompanyComponent } from './components/update-company/update-company.component';
import { HttpClientModule } from '@angular/common/http';
import { environment } from 'src/environments/environment';
import { InMemoryWebApiModule } from 'angular-in-memory-web-api';
import { CompanyInMemDataService } from './services/company-in-mem-data';
import { ReactiveFormsModule } from '@angular/forms';
import { HttpClientCompanyService } from './services/http-client-company.service';

@NgModule({
  declarations: [
    AppComponent,
    ListCompaniesComponent,
    CreateCompanyComponent,
    UpdateCompanyComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    environment.production ?
      [] : InMemoryWebApiModule.forRoot(CompanyInMemDataService),
    ReactiveFormsModule

  ],
  providers: [HttpClientCompanyService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

The main change to app.module.ts is the code shown below:

TypeScript

```typescript
environment.production ?
      [] : InMemoryWebApiModule.forRoot(CompanyInMemDataService),
```

- This is a switch that ensures the in-memory-web-api will be used in non-production environments.