V19 Sharing DI Services

Here are some steps to include, when creating a public library that uses services from DI, such as HttpClient.

We cater the example on this page to using HttpClient.
But, they apply, equally, to any library that consumes from DI.

Library Service

Our example is a library service that uses HttpClient.
It will inject HttpClient from DI.

NOTE: Our service example will require that something registers HttpClient with DI, so that no error occurs at runtime.
So, we have a providers instance, below, that needs to be run by any consuming app.
Think of it as an initialization for the library.

Here's what the library service looks like:

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

@Injectable({
  providedIn: 'root'
})
export class MyLibraryService {
  private http = inject(HttpClient);

  constructor() {}

  getData() {
    return this.http.get('https://api.example.com/data');
  }
}
```

Note that the above example injects the HttpClient, directly, into a private variable, instead of in the constructor.

This is a new injection method, allowed in V18.

Library Providers

Next, our library needs to include a providers instance that any consuming library or app can add to its providers array.

For this, we create a file, lib-providers.ts, in the src folder of our library, that looks like this:

```
import { ApplicationConfig } from '@angular/core';
import { provideHttpClient, withFetch } from '@angular/common/http';

export const LIB_OGAWEBUISHAREDKERNEL_PROVIDERS: ApplicationConfig =
{
  providers: [provideHttpClient(withFetch())]
};
```

Note: We've set the provider name to be unique to our library.
You, as well, will want to do this, to prevent colliding with the providers entry of another library.

Library Export

The public-api.ts file of our library needs to include an export statement for the providers instance, above:

```
export * from './lib-providers';
```

Consuming Library

Any library that consumes our library needs to add our provider to its providers array, like this lib-providers.ts:

```
import { ApplicationConfig } from '@angular/core';
import { LIB_OGAWEBUISHAREDKERNEL_PROVIDERS } from 'my-library';

export const CONSUMER_LIB_PROVIDERS: ApplicationConfig = {
  providers: [
    ...LIB_OGAWEBUISHAREDKERNEL_PROVIDERS.providers // □ Inherit HttpClient from `my-library`
  ]
```

```
};
```

The above will ensure that any consumer of our consuming library, will also include the providers from our library.

Consuming Service

Any service in our consuming library or app that consumes the service from our library, will inject it, like this:

```
import { Injectable, inject } from '@angular/core';
import { MyLibraryService } from 'my-library';

@Injectable({
  providedIn: 'root'
})
export class ConsumerLibraryService {
  private myLibraryService = inject(MyLibraryService);

  fetchData() {
    return this.myLibraryService.getData(); // □ Calls the service from `my-library`
  }
}
```

Consuming Library Public-API.ts

The public-api.ts file of the consuming library, will include a similar export of its providers instance, so that any registrations are performed by consumers of it.

That public-api.ts includes these elements:

```
export * from './consumer-service';
export * from './providers'; // □ Expose CONSUMER_LIB_PROVIDERS for easy consumption
```

Consuming Application

And, any application that consumes our library, or the intermediate consuming library, needs to import it and add its providers to the app's bootstrap statement, like this:

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app/app.component';
import { CONSUMER_LIB_PROVIDERS } from 'consumer-library';

bootstrapApplication(AppComponent, CONSUMER_LIB_PROVIDERS);
```

> NOTE: If your consuming app does NOT register the DI provider of the library, you will see a runtime error, that the injector cannot resolve the HttpClient.

Consuming Component

This leaves our last example of a component of our app, that consumes the service from our library.

For a pre-V19 app, it looks like this:

```
import { Component, inject } from '@angular/core';
import { ConsumerLibraryService } from 'consumer-library';

@Component({
  selector: 'app-root',
  template: `<button (click)="loadData()">Fetch Data</button>`,
})
export class AppComponent {
  private consumerService = inject(ConsumerLibraryService);

  loadData() {
    this.consumerService.fetchData().subscribe(data => {
      console.log('Received Data:', data);
    });
  }
}
```

For a v19 and later app, the provider gets referenced in the app.config.ts file, like this:

```
import { ApplicationConfig, provideZoneChangeDetection } from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';
```

```
import { LIB_OGAWEBUISHAREDKERNEL_PROVIDERS } from '../../../lib-oga-webui-sharedkernel/src/lib-providers';


export const appConfig: ApplicationConfig = {
  providers:
  [
    provideZoneChangeDetection({ eventCoalescing: true }), provideRouter(routes),
    LIB_OGAWEBUISHAREDKERNEL_PROVIDERS.providers
  ]
};
```

Revision #4
Created 15 March 2025 22:45:35 by glwhite
Updated 16 March 2025 01:29:13 by glwhite