5.- Conceptos avanzados de Angular

5.1.- Módulos

5.1.1.- NgModule API

```
@NgModule({
  // Static, that is compiler configuration
  declarations: [], // Configure the selectors
  entryComponents: [], // Generate the host factory

  // Runtime, or injector configuration
  providers: [], // Runtime injector configuration

  // Composability / Grouping
  imports: [], // composing NgModules together
  exports: [] // making NgModules available to other parts of the app
})
```

Propiedad	Descripción
declarations	Clases declarables (componentes, directivas y *pipes*). Todas estas clases se declaran en un solo módulo, según necesitemos, si no dará fallo el compilador.
providers	Clases inyectables (*Services* y algunas clases). Angular tiene asociado a cada módulo un injector, por lo que el módulo raíz tiene su injector raíz y cada submódulo tiene el suyo propio que parte del principal.
imports	Todos aquellos módulos o librerías de Angular que se vayan a utilizar en la aplicación.
exports	Clases declarables (componentes, directivas y *pipes*) que quieran ser importadas por otro módulo
bootstrap	Componente de arranque de la aplicación
entryComponents	Componentes que pueden ser cargados dinámicamente en la vista

5.1.2.- Multimodulado o submódulos

Angular al ser un *framework* que trabaja de forma modular, tiene la posibilidad de dividir sus aplicaciones en diferentes áreas, por ejemplo por su funcionalidad, y así aligerar la primera carga de la web.

Esta división se hace a través de submódulos. En cada uno de estos módulos cargaremos todo lo necesario para esa área. Con esta forma de trabajar todos estos ficheros no se cargarán hasta que no accedemos a esa área de trabajo.

Esta forma de trabajo también es bastante útil para organizar nuestro proyecto si alcanza un tamaño considerable.

Para crear un nuevo módulo usaremos el siguiente comando:

```
ng generate module nameModule
```

Esto nos generará el siguiente código:

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

@NgModule({
  imports: [
    CommonModule
  ],
  declarations: []
})
export class CustomerDashboardModule { }
```

Como se puede ver es exactamente igual que el módulo principal de una aplicación en Angular. Entonces ¿cómo va a saber Angular cual es el módulo principal? Para eso es el archivo *main.ts* que se explica al principio del documento y que le 'dice' a Angular cual es el módulo principal a cargar.

```
platformBrowserDynamic().bootstrapModule(AppModule).catch(err => console.error(err));
```

Una vez creado el módulo podemos añadirle cualquier elemento (componente, directiva, services, etc) usando el Angular CLI.

Si hemos creado un módulo cuyo archivo se llama *customer-dashboard.module.ts* entonces el comando a introducir será:

```
ng generate component customerDashboard
```

Al crear el componente Angular irá buscando desde la ruta donde nos encontremos hacia la raíz del proyecto un módulo al que importar el componente hasta llegar al módulo principal e importarlo ahí

Una vez creado el nuevo módulo tan solo queda importalo en el módulo principal y añadirlo a *imports* para poder usarlo en la aplicación.

¿Cómo visualizar estos nuevos componentes de submódulos?

Existen dos formas de mostrar los componentes añadidos a submódulos:

- Añadiendo el componente a los *exports* del submódulo y así poder utilizar su selector en otro componente.
- Añadiendo rutas al submódulo y sus componentes.

El primer método tan solo se necesita añadir el componente del submódulo al array de *exports* de este mismo submódulo. De esta forma podemos usar su selector para cargar ese componente en otro lado de la aplicación, como por ejemplo en el AppComponent.

Para el segundo caso debemos seguir estos pasos:

1. Añadir la ruta al *app-routing*.

```
{ path: 'customer', loadChildren: '.. rutadelarchivo/customer-dashboard.module#CustomerDashboarModule' }
```

2. Crear en el *imports* del submódulo las subrutas para 'llegar' hasta el componente.

```
…
imports: [
  …
  RouterModule.forChild([
    { path: "", component: CustomerDashboardComponent }
  ]),
  …
]
...
```

5.1.3.- Providers

Providers es una instrucción del Inyector de Dependencias para obtener el valor de una dependencia. Esto quiere decir que todos los archivos que Angular detecte que se almacenan en el ID deben de ser importados en el *providers* para poder ser utilizados. La mayoría de estos archivos suelen ser *services*.

Dependiendo del nivel donde importemos ese archivo en el *providers* correspondiente podremos usarlo en toda o solo parte de la aplicación:

- Toda la aplicación:
    - El archivo se importará en el *providers* del módulo principal de nuestra aplicación.

    - @Injectable({ providedIn: ‘root’ }) Cuando un archivo, normalmente un *service,* tiene este metadata significará que Angular cargará ese archivo en el providers principal sin necesidad de tener que hacer el punto anterior.
- Para un solo submódulo cuando tenemos multimodulado:
    - En el *providers* de este submódulo.
    - @Injectable({ providedIn: nombreModulo }) donde nombreModulo será la clase del submódulo.

```
import { Injectable } from '@angular/core';
import { UserModule } from './user.module';

@Injectable({
  providedIn: UserModule,
})
export class UserService {
}
```

- Para un solo componente:
    - En el *providers* de ese componente.

5.2.- *Routing*

Hasta ahora se había visto como crear rutas sencillas entre componentes. Pero se pueden hacer varios niveles de enrutado para una aplicación web.

```
const crisisCenterRoutes: Routes = [
  {
    path: 'crisis-center', component: CrisisCenterComponent, children: [
      {
        path: '', component: CrisisListComponent,  children: [
          {
            path: ':id',  component: CrisisDetailComponent
          },
          {
            path: '', component: CrisisCenterHomeComponent
          }
        ]
      }
    ]
  }
];
```

Pero no basta con esto, no se debe olvidar colocar la etiqueta *router-outlet* en cada componente que tenga rutas ‘hijo’ para que Angular las reconozca y sepa donde renderizarlas.

CanActivate

Se utiliza para limitar a los usuarios el acceso a las rutas que nosotros no queramos que accedan, por ejemplo, una zona de administración de una web.

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';

@Injectable({
  providedIn: 'root',
})
export class AuthGuard implements CanActivate {
  canActivate( next: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    // Código para comprobar el token o clave para devolver true o false. Lo habitual justo antes de devolver
false se reenvie con la clase router de Angular a otra vista, si no la app se quedará parada en este punto.
  }
}
```

Una vez creado el código anterior se debe de implementar en el array de rutas de la aplicación.

```
{ path: 'categorias', canActivate: [AuthGuard], component: 'CategoriasComponent' }
```

Eventos

Durante la navegación, el objeto *Router* emite una serie de eventos a través de la propiedad *Router.events*.

```
this.router.events.subcribe( (event) => {
  console.log(event);
});
```

Algunos de esos eventos son:

NavigationStart	NavigationEnd	NavigationError
GuardsCheckEnd	ChildActivationStart	ResolveStart

Para saber la lista entera pulse aquí.