3. Modules

Apparté : Modules ES6

** Les modules ES6 sont différents et ne porte pas le même objectif qu'un Module Angular. **

Les modules ES6 sont issus de la Norme Ecmascript, et permettent de regrouper du code dans un fichier Javascript. Dans ces fichiers, le développeur choisit d'importer et / ou d'exporter différentes propriétés : classes, méthode, variables. Ces propriétés peuvent ensuite être importées et utilisées dans un autre fichier.

Exemple:

- Export de fonctions :

```
// fichier test.exp.js
export default function maFonctionParDefaut(number) {
  return "Hello " + number;
}

export function maFonctionPlip() {
  return "Hello Plip";
}
```

- Import des fonctions

```
import fonctionDefaut from "./operation.js";
import { maFonctionPlip } from "./operation.js";

console.info("default ", fonctionDefaut(10));
console.info("plip ", maFonctionPlip());
```

- En sortie

> default Hello 10
> plip Hello Plip

Plus d'information sur la différence entre un module ES6 et un NgModule Angular

Modules Angular

Un module Angular regroupe un ensemble de fonctionnalités :

- composants graphiques,
- appels à des services,
- formatage de données,
- dépendances vers d'autres modules,
- etc.

Les modules permettent donc de découper une application en plusieurs sous blocs logiques. L'intérêt est multiple : découpage logique d'une application, meilleure comprehension du découpage d'un projet, optimisation des temps de chargement, etc.

Décorateur @NgModule

Le décorateur @NgModule est nécessaire pour décrire le module Angular à packager. Celui-ci est défini en entête de classe :

```
@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Un module Angular contient les méta données suivantes :

- `bootstrap` : Composant `root` qui sera utilisé pour générer le module,
- `declarations` : Tous les composants, directives, pipes exploités dans les templates HTML doivent être référencés ici,
- `entryComponents` : Tous les composants qui ne sont pas directement référencés dans les templates HTML, mais qui sont potentiellement instanciés dans un contrôleur, ou une autre classe.
- `exports` : Liste des composants visible par un module qui exploiterait ce module,
- `imports` : Tous les modules dont dépend le module à exporter,
- `providers` : variables injectées par angular, qui pourront être récupérer via l'injecteur, ou de manière automatique dans les constructeur. Voir le chapitre sur les injections de dépendances,
- `schemas` : Les éléments et propriétés qui ne sont pas des components ou directive Angular qui sont à déclarer dans le schéma des MétaDonnées. Les valeurs possibles sont NO_ERRORS_SCHEMA et CUSTOM_ELEMENTS_SCHEMA. Cela permet d'intégrer des composants non basés sur Angular intégrant un – dans le nom de la template.

Documentation officielle

Exercice

Présentation

Application Météo.

L'objectif de cet excercice est de créer un premier module dans l'application Angular.

Etapes de création d'un module

Nous allons tout d'abord créer l'application avec le CLI.

1. Créer une application `web-meteo` à l'aide du CLI Angular. (voir chapitre précédent) Evitez d'importer le routeur directement, nous le ferons manuellement dans un prochain chapitre.

> ng new web-meteo

2. Nettoyer l'application pour qu'il n'y ait plus qu'un seul composant `app.component`, avec une templater vide.
3. Dans le composant existant `app.component`, ajouter le le titre de l'application "Ma Météo".
4. Lancer l'application et vérifier que l'application s'affiche correctement avec le titre.

> ng serve

5. Créer un module `meteo.module.ts`.

> ng generate module meteo

6. Importer ce module dans le module principal de l'application via la propriété `imports`.

Vous venez de créer votre 1er module Angular ! Il est pour le moment vide, mais nous allons très rapidement y ajouter des composants.

Correction disponible ici