

Host binding et Host listener

Host listener

- Retour sur le composant folder :

```
<div class= "box" (click)="onClick($event)">
  {{ tree.value }}

  <ng-container *ngIf="expanded">
    <app-tree *ngFor="let child of tree.children" [tree]="child" />
  </ng-container>
</div>
```

- Angular génère un élément dans le HTML pour chaque composant, et ici un div qui encapsule tout le composant

```
▼ <app-root ng-version="17.0.5"> == $0
  ▼ <app-tree expanded="true" _ngghost-ng-c135607009 ng-reflect-expanded="true" ng-reflect-tree="[object Object]">
    ▼ <div _ngcontent-ng-c135607009 class="box">
      " root "
      ▶ <app-tree _ngcontent-ng-c135607009 _ngghost-ng-c135607009 ng-reflect-tree="[object Object]">...</app-tree>
      ▶ <app-tree _ngcontent-ng-c135607009 _ngghost-ng-c135607009 ng-reflect-tree="[object Object]">...</app-tree>
```

- On a déjà vu qu'il est possible de spécifier du style directement sur les composants avec le sélecteur :host
- De manière similaire, on peut écouter des événements directement sur l'élément du composant
- Pour cela on utilise la propriété host dans le décorateur @Component

```
@Component({  
  ...  
  host: {  
    '(click)': 'onClick($event)'  
  }  
})
```

- Il est aussi possible d'utiliser le décorateur @HostListener directement sur la méthode dans le composant

```
@HostListener('click', ['$event'])  
onClick(event: Event) {
```

- La préconisation d'Angular est d'utiliser host plutôt que le décorateur @HostListener

Exercice

- Refaire le composant tree-composant en enlevant le div global

Host binding

- Il est également possible de lier des propriétés et les attributs à l'élément hôte du composant
- Pour cela on utilise également la propriété host

```
@Component({  
  ...  
  host: {  
    '[style.font-size.px]': 'fontSize',  
    '[style.color]': 'color'  
    '[attr.disabled]': 'disabled'  
  }  
})
```

- Ou le décorateur @HostBinding

```
@HostBinding('style.font-size.px')  
fontSize = 60  
  
@HostBinding('style.color')  
color = 'red'
```

Organisation d'un projet Angular

Bonnes pratiques SNCF

- <https://dev.sncf/docs/frontend/angular/>
- Les bonnes pratiques présentées sont celles de la SNCF, pas des vérités absolues !

Création d'un projet

```
ng new --skip-git --inline-style --inline-template --directory ./ --routing --skip-tests --style scss {nom-du-projet}-cli
```

inline-style et inline-template

- Lors de la création d'un nouveau composant avec `ng generate`, le style et le template sont dans le même fichier
- La préconisation de la SNCF est de faire des single file component

skip-tests

- `ng generate` ne génère pas les fichiers de tests

ng generate

- Permet de générer et modifier automatiquement les fichiers du projet Angular pour ajouter des nouveaux éléments (Module, Composant, ...)
- Nomme automatiquement les classes et les fichiers en suivant les bonnes pratiques
- <https://angular.io/cli/generate>

Organisation d'une application

- `src/`
 - `app/` -- Composants et services nécessaires au fonctionnement de l'application
 - `assets/` -- Contient les images, fonts, ressources de l'application
 - `config/` -- Fichier(s) de configuration qui seront utilisés par la shared lib de configuration
 - `environments/`
 - `features/` -- Contient les features modules
 - `generated/` -- Contient tous les fichiers générés par des outils de génération de code
 - `shared/` -- Contient tout ce qui est partagé à travers l'application
 - `styles/`

app/

services/

components/

pipes/

.../

app.module.ts

app-routing.module.ts

- Point d'entrée de l'application
- Contient les éléments nécessaires au démarrage de l'application

App module

- Importe d'autres features modules
- Importe le shared module ou des éléments standalone si besoin
- N'exporte rien

features/

```
feature/  
  services/  
  components/  
  .../  
  feature.module.ts  
  feature.routing.module.ts  
other-feature/  
  services/  
  components/  
  .../  
  other-feature.module.ts  
  other-feature.routing.module.ts
```

- Chaque feature représente une partie de l'application

Feature module


- Le seul export d'un feature module est son composant racine
- Importe le shared module ou des éléments standalone si besoin
- Importes d'autres features modules

shared/

```
components/  
enums/  
types/  
services/  
utils/  
...  
(shared.module.ts)
```

- Contient les éléments réutilisables de l'application
- Les composants réutilisés sont de bon candidats pour être standalone, ce qui évite de faire un `shared.module`

Shared module

- Exporte tous les éléments réutilisables de l'application
-  Ne pas fournir de providers dans le shared module

environments/

environment.ts

environment.development.ts

- Contient les fichiers spécifiques à un environnement (généralement des constantes)
- Il est possible de générer les fichiers et la configuration associée avec la commande

```
ng generate environments
```

- Exemple de fichier environment.ts

```
export const environment = {  
  production: true,  
};
```

- Et le fichier environment.development.ts associé

```
export const environment = {  
  production: false,  
};
```

environments/

- Pour l'utiliser, on importe le fichier original dans le reste du programme

```
import { environment } from '../environments/environment';
```

```
console.log(environment.production);
```

- Le fichier sera remplacé selon l'environnement