

COMPOSANTS, NAVIGATION MODULES, SERVICE

INF1013, Hiver 2021

DMI, UQTR

PLAN

- Cycle de vie des composants
- Composants Dynamiques
- Services de données
- Intégration de Material Design

Cycle de vie des composants: Introduction

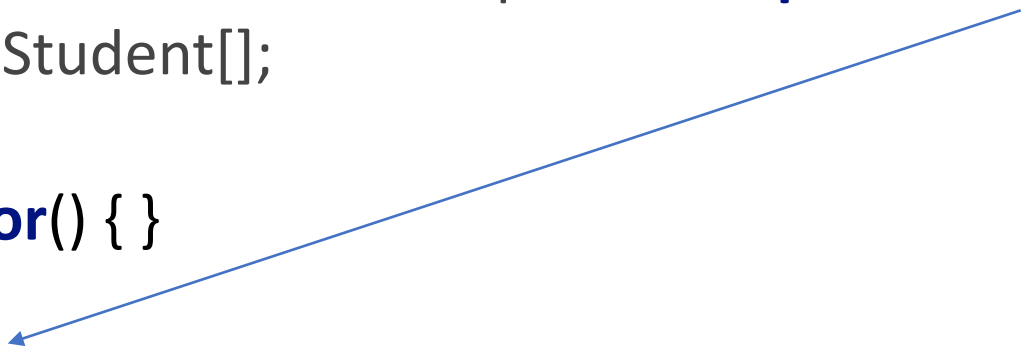
- Les composants sont initialisés par inversion de contrôle,
 - Ils sont gérés par Angular,
 - Ils ont donc un cycle de vie.
- Pour tout composant, Angular fait la création, le rendu en même temps que les enfants (du composant), s'occupe de la liaison des données avec le template et le détruit avant de le retirer du DOM.
- Angular offre une série de méthodes de rappel lors de ces différentes étapes de la vie du composant.
- Les directives étant des composants, sont donc soumis au même concept de cycle de vie.

Cycle de vie des composants: Organisation

- Angular organise ces méthodes de rappels dans différentes interfaces.
- Le développeur implémente ces interfaces.
 - Il redéfinit les méthodes de cycle de vie qui l'intéressent
 - Chaque interface a une méthode de rappel unique.
 - Le nom de la méthode est le nom de l'interface préfixé par ng.
- Par exemple, pour utiliser les méthode d'initialisation,
 - L'interface [OnInit](#) contient la méthode de rappel `ngOnInit()`
 - Cette méthode est appelée juste après la création du composant.

Cycle de vie des composants: Exemple

```
export class ListStudentsComponent implements OnInit {  
  students: Student[];  
  
  constructor() { }  
  
  ngOnInit() {  
    this.students = [  
      new Student('Michel', 'Leblanc', 'leblancmi432', 90),  
      new Student('Isabelle', 'Gagné', 'gagnisab4333', 98),  
    ];  
  }  
}
```



Cycle de vie des composants: Séquences

Méthode de Rappel	Objectif et Séquence
ngOnChanges()	Appelé avant ngOnInit() et si un ou plusieurs plusieurs donnée bindées changent
ngOnInit()	Appelée <i>une fois</i> , après le premier ngOnChanges().
ngDoCheck()	Detect and act upon changes that Angular can't or won't detect on its own. Called during every change detection run, immediately after ngOnChanges() and ngOnInit().
ngAfterContentInit()	Respond after Angular projects external content into the component's view / the view that a directive is in. Called <i>once</i> after the first ngDoCheck().
ngAfterContentChecked()	Respond after Angular checks the content projected into the directive/component Called after the ngAfterContentInit() and every subsequent ngDoCheck().
ngAfterViewInit()	Respond after Angular initializes the component's views and child views / the view that a directive is in. Called <i>once</i> after the first ngAfterContentChecked().
ngAfterViewChecked()	Respond after Angular checks the component's views and child views / the view that a directive is in. Called after the ngAfterViewInit() and every subsequent ngAfterContentChecked().
ngOnDestroy()	Cleanup just before Angular destroys the directive/component. Unsubscribe Observables and detach event handlers to avoid memory leaks. Called <i>just before</i> Angular destroys the directive/component.

Cycle de vie des composants: OnInit

- *OnInit()* doit s'utiliser pour deux raisons principales:
 - Faire des initialisations complexes juste après la construction
 - Configurer le composant après initialisation des propriétés.
- La construction d'un composant doit rester simple et fonctionnelle.
- Le constructeur ne doit faire qu'initialiser les variables membre.
- Privilégier OnInit pour récupérer les données, observer les variable...

Cycle de vie des composants: OnDestroy

OnDestroy() traite la logique de libération de ressources qui doit se faire avant que Angular ne collecte ce composant.

C'est le moment aussi de notifier d'autres parties de l'Application de la destruction du composant si besoin est.

Exemple:

- Dés-enregistrement des Observables et des événements du DOM.

- Arrêter les *timers*.

- Ne pas libérer les ressources peut conduire à des fuites de mémoire.

Navigation: Principale et Secondaire.

- Dans notre applications Université, nous souhaitons ajouter un menu de navigation principale (barre de navigation) avec comme item
 - Notre université
 - Nos programmes
 - Nos cours
- Dans la liste des programmes et des cours.
 - L'ouverture d'un cours ouvre les details de celui-ci.
 - L'ouverture d'un programme ouvre les details de ce dernier.

Routes Définitions et Implémentation

- Les routes permettent à la navigation de se faire entre les composants.
- La navigation permet un changement dynamique du contenu d'un slot à l'aide de la barre de navigation du navigateur/ancres/bouton/autres.
- Ainsi sont définies les Routes.

```
const routes: Routes = [  
  { path: 'first-component', component: FirstComponent },  
  { path: 'second-component', component: SecondComponent },  
];
```

Composants ciblé par le chemin

Chemins dans la barre de navigation

Routes: Importation

- Une fois les routes définies, elles doivent être importée dans l'application.
 - Soit dans le module AppModule
 - Soit dans un module de route, séparé
 - Qui sera ensuite importé dans AppModule
- Création de module de route s'il n'a pas déjà intégré lors de la création du projet
 - `ng new routing-app --routing`
 - Cela créera le module RoutingApp et l'importera dans le module principal AppModule

Route: Le Module de Routage

```
import { NgModule } from '@angular/core';  
import { Routes, RouterModule } from '@angular/router';
```

Déclaration des routes

```
const routes: Routes = [...];
```

```
@NgModule({  
  imports: [RouterModule.forRoot(routes)],  
  exports: [RouterModule]  
})
```

Importation des routes

Exporter le nouveau Module pour qu'il soit importable

```
export class AppRoutingModule { }
```

Modules de Routage

Routes: Outlets et RouterLink

- Les outlets sont des composants de la navigation.
- Les outlets sont les slots dans lesquels seront placés les composants navigables.
 - Les outlets sont placés dans le composant hôte (parent) nous forme de balises
- Les routerLinks sont le ancre qui sont la source de la navigation.
 - Les routerLinks se présentent sous forme de directives.

```
<router-outlet></router-outlet>
```

```
<a [routerLink]="/first-component" routerLinkActive="active">Ouvrir first Component</a>
```

Routes: Quelques propriétés

- D'autres propriété des routes existes, elles entre autres
 - La redirection
 - La gestion des chemins inexistants (chemins alias)
 - Les chemin par défauts

```
const routes: Routes = [  
  { path: 'first-component', component: FirstComponent },  
  { path: 'second-component', component: SecondComponent },  
  { path: '', redirectTo: '/first-component', pathMatch: 'full' }, // redirect to `first-component`  
  { path: '**', component: PageNotFoundComponent }, // Wildcard route for a 404 page  
];
```

Routes: Notes importantes

- L'ordonnancement des routes lors de leur déclaration a toute son importance !!!
 - Elle obéit à la stratégie du Fifo.
- Il est alors important de définir
 - les chemins avec nom explicites (statics) en premier
 - Les chemins vide (défauts)
 - Puis les chemins génériques à la fin

Routes: Passage de paramètres

- Dans la plus part des cas, on souhaite, lors de la navigation, passer des information de la source (hôte du routerLink) (S) vers les composant navigable (N).
- Ici, impossible d'utiliser @Output car
 - Le composant hôte (H) du router-outlet n'est pas nécessairement le parent
 - Le sélecteur du composant(N) ne figure pas statiquement dans (H)
- Les passage d'information (envoi d'ID en général) se fait par les routes.

Routes: Passage de paramètres

- Pour passer des paramètres de S à N, on doit revoir la déclaration de la route en conséquence

```
const routes: Routes = [  
  { path: 'first-component/:param1', component: FirstComponent },  
  { path: 'second-component/:param2', component: SecondComponent },  
  { path: '**', component: PageNotFoundComponent }, // Wildcard route for a 404 page  
];
```

- Le binding expression des routeurLinks de S prendra la notation de [...] pour tenir compte des paramètres.

```
<a [routerLink]="['/details',selIdx]" routerLinkActive="active">Détails</a>
```

Routes: Passage de paramètres

- Comment N peut récupérer les paramètres envoyés par S?
- Pour se faire :
 - Nous injectons le composant `ActivatedRoute` dans N.
 - N implémente `OnInit` et va observer l'instance `paramMap` de l'instance de `ActivatedRoute`.
- la propriété `paramMap` est un/une `Observable`
 - Ainsi N peut réagir au changement de valeur de `paramMap` ie des paramètres de la route.
- Une dernière consistera à déplacer les données dans une source commune
 - Ce sont les services

Routes: Observation des paramètres

```
export class DetailsComponent implements OnInit {  
  student: Student;  
  constructor(private route: ActivatedRoute, private service: StudentDataService) {  
  }  
  
  ngOnInit(): void {  
    this.route.paramMap.subscribe(params => {  
      const idx = Number(params.get('idx'));  
      this.student = this.service.students[idx];  
    });  
  }  
}
```

Injection de dépendance

```
const routes: Routes = [{ path: 'details/:idx', component: DetailsComponent },...];
```

Routes: Composants Annexes Router/Guard

- Ce composant permet aussi de naviguer programmatiquement.
- Pour se faire on appelle la méthode `navigate(...)`

```
this.router.navigate(['/details', this.index]);
```

- Les guard permettent de limiter les navigations sous certaines condition [voir](#)

Intégration de Material Design

- Introduction au Material Design
 - <https://material.angular.io/guide/getting-started>
- Exemple d'utilisation de material List
 - <https://material.angular.io/components/list/api>

Exercice d'application

- Reprendre le lab précédent
 - créer un composant pour gérer la liste des étudiants
 - Déplacer le code nécessaire de app vers ce composant
 - Créer un service avec ng g s nom-du-service
 - déplacer la génération des données mock dans le service
 - Injecter le service dans le composant liste
 - Créer 3 composants.
 - 1 pour afficher les détails d'étudiant (réutiliser le composant)
 - 1 pour éditer un étudiant
 - 1 PageNotFound
 - Définir les routes pour la navigation
 - Définir le module de routing s'il n'est pas déjà défini
 - Passer l'index de la liste comme paramètre de navigation vers Details et Edit
 - Définir un Guard Evitant la navigation vers détails de étudiants ayant eu échec