Interface Système De L'Application (Shell)

Installer l'Interface Angular CLI

Installez l'Interface Angular CLI si vous ne l'avez pas déjà fait.

```
npm install -g @angular/cli
```

Créer Une Nouvelle Application

Créez un nouveau projet nommé angular-tour-of-heroes avec cette commande CLI.

```
ng new angular-tour-of-heroes
```

L'interface CLI Angular a généré un nouveau projet avec une application par défaut et des fichiers de support.

Servir L'Application

Allez dans le répertoire du projet et lancez l'application.

```
cd angular-tour-of-heroes
ng serve --open
```

*La commande **ng serve** crée l'application, démarre le serveur de développement, surveille les fichiers source et reconstruit l'application lorsque vous apportez des modifications à ces fichiers.*

*L'indicateur **–open** ouvre un navigateur à **http: // localhost: 4200 /**.*

Vous devrez voir l'application s'exécuter dans votre navigateur.

Composants Angular

La page que vous voyez est le shell de l'application. Le shell est contrôlé par un composant Angular nommé AppComponent.

Les composants sont les blocs de construction fondamentaux des applications angulaires. Ils affichent des données sur l'écran, écoutent les entrées de l'utilisateur et agissent en fonction de cette entrée.

Changer Le Titre De L'application

Ouvrez le projet dans votre éditeur ou IDE préféré et accédez au dossier src / app. Vous trouverez l'implémentation du shell AppComponent répartie sur trois fichiers:

1. app.component.ts—le code de la classe du composant, écrit en TypeScript.
2. app.component.html—le modèle de composant, écrit en HTML.
3. app.component.css—les styles CSS privés du composant.

Ouvrez le fichier de classe de composants (app.component.ts) et remplacez la valeur de la propriété title par « Tour of Heroes ».

app.component.ts (class title property)

```
title = 'Tour of Heroes';
```

Ouvrez le fichier de modèle de composant (app.component.html) et supprimez le modèle par défaut généré par la CLI Angular. Remplacez-le par la ligne de HTML suivante.

app.component.html (template)

```
<h1>{{title}}</h1>.
```

Les doubles accolades sont la syntaxe de liaison d'interpolation d'Angular. Cette liaison d'interpolation présente la valeur de la propriété `title` du composant dans la balise d'en-tête HTML.

Le navigateur actualise et affiche le nouveau titre de l'application.

Ajouter Les Styles D'Application

La plupart des applications recherchent un aspect cohérent dans l'application. L'interface de ligne de commande a généré un styles.css vide à cet effet. Mettez y vos styles à l'échelle de l'application.

Voici un extrait du fichier `styles.css` pour l'application exemple Tour of Heroes.

src/styles.css (excerpt)

```
/* Application-wide Styles */
h1 {
  color: #369;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 250%;
}
h2, h3 {
  color: #444;
  font-family: Arial, Helvetica, sans-serif;
  font-weight: lighter;
}
body {
  margin: 2em;
}
body, input[text], button {
  color: #888;
  font-family: Cambria, Georgia;
}
/* everywhere else */
* {
  font-family: Arial, Helvetica, sans-serif;
}
```

Examen Final Du Code

Le code source de ce tutoriel et les styles globaux de Tour of Heroes sont disponibles dans l'exemple en direct / exemple de téléchargement.

Voici les fichiers de code discutés sur cette page.

src/app/app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Tour of Heroes';
}
src/app/app.component.html


<h1>{{title}}</h1>
src/styles.css (excerpt)
```

```
/* Application-wide Styles */
h1 {
  color: #369;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 250%;
}
h2, h3 {
  color: #444;
  font-family: Arial, Helvetica, sans-serif;
  font-weight: lighter;
}
body {
  margin: 2em;
}
body, input[text], button {
  color: #888;
  font-family: Cambria, Georgia;
}
/* everywhere else */
* {
  font-family: Arial, Helvetica, sans-serif
```

Récapitulons! Jusqu'à maintenant nous avons pu :

- créer la structure d'application initiale à l'aide de l'interface CLI angulaire.
- apprendre que les composants Angular affichent les données.
- utiliser les doubles accolades d'interpolation pour afficher le titre de l'application.

L'application a maintenant un titre de base. Ensuite, nous allons créer un nouveau composant pour afficher les informations sur les héros et placer ce composant dans le shell de l'application.

L'Éditeur Des Héros

Créer Le Composant Des Héros

À l'aide de l'interface de ligne de commande Angular CLI, générez un nouveau composant nommé `heroes` (héros).

```
ng generate component heroes
```

La CLI crée un nouveau dossier, `src / app / heroes /` et génère les trois fichiers de `HeroesComponent`.

Le fichier de classe `HeroesComponent` est le suivant:

app / heroes / heroes.component.ts (version initiale)

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
})
export class HeroesComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

Vous devez toujours importer le symbole de composant à partir de la bibliothèque de base Angular et annoter la classe de composant avec @Component.

```
hero = 'Windstorm';
```

Montrer Le Héros

Ouvrez le fichier de modèle heroes.component.html. Supprimez le texte par défaut généré par l'interface de ligne de commande angulaire et remplacez-le par une liaison de données à la nouvelle propriété hero.
heroes.component.html

```
{{hero}}
```

Afficher La Vue Du Composant Héros

Pour afficher le HeroesComponent, vous devez l'ajouter au modèle du shell AppComponent.

Rappelez-vous que l'app-heroes est le sélecteur d'élément pour HeroesComponent. Ajoutez donc un élément <app-heroes> au fichier modèle AppComponent, juste en dessous du titre.

rc / app / app.component.html

```
<h1>{{title}}</h1>
<app-heroes></app-heroes>
```

Pour afficher le HeroesComponent, vous devez l'ajouter au modèle du shell AppComponent.

c/app/app.component.html

```
<h1>{{title}}</h1>
<app-heroes></app-heroes>
```

Assuming that the CLI ng serve command is still running, the browser should refresh and display both the application title and the hero name.

Créer Une Classe De Héros

Un vrai héro est plus qu'un nom.

Créez une classe Hero dans son propre fichier dans le dossier src / app. Donnez-lui les propriétés de l'id et du nom.

src/app/hero.ts

```
export class Hero {
  id: number;
  name: string;
}
```

Revenez à la classe HeroesComponent et importez la classe Hero.

Refactoriser la propriété hero du composant pour qu'elle soit de type Hero. Initialisez-le avec un ID de 1 et le nom Windstorm.

Le fichier de classe HeroesComponent révisé devrait ressembler à ceci:

src/app/heroes/heroes.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Hero } from '../hero';

@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
```

```
})
export class HeroesComponent implements OnInit {
  hero: Hero = {
    id: 1,
    name: 'Windstorm'
  };

  constructor() { }

  ngOnInit() {
  }

}
```

La page ne s'affiche plus correctement car vous avez changé hero d'une chaîne (string) en un objet.

Afficher L'objet Hero

Mettez à jour la liaison dans le modèle pour annoncer le nom du héros et afficher à la fois l'identifiant id et le nom name dans une disposition détaillée comme ceci:

heroes.component.html (HeroesComponent's template)

```
<h2>{{ hero.name }} Details</h2>
<div><span>id: </span>{{hero.id}}</div>
<div><span>name: </span>{{hero.name}}</div>
```

Le navigateur actualise et affiche les informations du héros.

Format Avec La MajusculePipe *UppercasePipe*

Modifiez la liaison hero.name comme ceci.

```
<h2>{{ hero.name | uppercase }} Details</h2>
```

Le navigateur se rafraîchit et maintenant le nom du héros est affiché en majuscules.

Le mot majuscule uppercase dans la liaison d'interpolation, juste après l'opérateur de tuyau (|), active le majuscules intégré UppercasePipe.

Les pipes sont un bon moyen de formater les chaînes strings, les montants monétaires, les dates et autres données d'affichage. Angular est livré avec plusieurs pipes intégrés et vous pouvez créer les votres.

Modifier Le Héro

Les utilisateurs doivent pouvoir modifier le nom du héros dans une zone de texte <input>.

La zone de texte doit à la fois afficher la propriété de nom du héros name et mettre à jour cette propriété en tant que type de l'utilisateur. Cela signifie que les données s'écoulent de la classe du composant vers l'écran et de l'écran vers la classe.

Pour automatiser ce flux de données, configurez une liaison de données bidirectionnelle entre l'élément de formulaire <input> et la propriété hero.name.

Liaison Bidirectionnelle

Refactorisez la zone de détails dans le modèle HeroesComponent afin qu'il ressemble à ceci:

src / app / heroes / heroes.component.html (modèle de HeroesComponent)

```
<div>
    <label>name:
      <input [(ngModel)]="hero.name" placeholder="name">
```

```
    </label>
</div>
```

[(ngModel)] est la syntaxe de liaison de données bidirectionnelle d'Angular

Ici, il attache la propriété hero.name à la zone de texte HTML afin que les données puissent circuler dans les deux sens: de la propriété hero.name à la zone de texte, et de la zone de texte à hero.name.

Le Module De Formulaires *FormsModule* Manquant

Notez que l'application a cessé de fonctionner lorsque vous avez ajouté [(ngModel)]. Pour voir l'erreur, ouvrez les outils de développement du navigateur et regardez dans la console pour un message comme

```
Template parse errors:
Can't bind to 'ngModel' since it isn't a known property of 'input'.
```

Bien que ngModel soit une directive Angular valide, elle n'est pas disponible par défaut.

Elle appartient à FormsModule, facultatif, et vous devez choisir de l'utiliser.

Module App *AppModule*

Angular doit savoir comment les éléments de votre application s'emboîtent et quels autres fichiers et bibliothèques l'application requiert. Cette information est appelée métadonnées

Certaines des métadonnées se trouvent dans les décorateurs @Component que vous avez ajoutés à vos classes de composants. Les autres métadonnées critiques sont dans les décorateurs @NgModule.

Le plus important ng module décorateur @NgModuledecorator annote la classe Module app AppModule de niveau supérieur.

L'interface de ligne de commande Angular CLI a généré une classe module app AppModule dans src / app / app.module.ts lors de la création du projet. C'est ici que vous vous inscrivez au module formulaire FormsModule.

Importer Les Modules Formulaires FormsModule

Ouvrez AppModule (app.module.ts) et importez le symbole FormsModule à partir de la bibliothèque @ angular / forms.

app.module.ts (FormsModule symbol import)

```
import { FormsModule } from '@angular/forms'; // <-- NgModel lives here
```

Ajoutez ensuite FormsModule au tableau imports de la métadonnée @NgModule, qui contient une liste de modules externes dont l'application a besoin.

app.module.ts (@NgModule imports)

```
imports: [
  BrowserModule,
  FormsModule
],
```

Lorsque le navigateur est actualisé, l'application devrait fonctionner à nouveau. Vous pouvez modifier le nom du héros et voir les changements reflétés immédiatement dans le <h2> au-dessus de la zone de texte.

Déclarez le composant Héros HeroesComponent

Chaque composant doit être déclaré dans exactement un NgModule

Vous n'avez pas déclaré le HeroesComponent. Alors pourquoi l'application a-t-elle fonctionné? Cela a fonctionné parce que l'interface de ligne de commande Angular CLI a déclaré le composant Héros HeroesComponent dans l'AppModule lorsqu'il a généré ce composant.

Ouvrez src / app / app.module.ts et trouvez HeroesComponent importé en haut.

```
import { HeroesComponent } from './heroes/heroes.component';
```

Le composant HeroesComponent est déclaré dans le tableau @NgModule.declarations.

```
declarations: [
  AppComponent,
  HeroesComponent
],
```

Notez que AppModule déclare les composants d'application, AppComponent et HeroesComponent.

La Revue Finale Des Codes

Votre application devrait ressembler à cet exemple direct / téléchargement de l'exemple. Voici les fichiers de code discutés sur cette page.

src/app/heroes/heroes.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Hero } from '../hero';

@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
})
export class HeroesComponent implements OnInit {
  hero: Hero = {
    id: 1,
    name: 'Windstorm'
  };

  constructor() { }

  ngOnInit() {
  }

}
```

src/app/heroes/heroes.component.html

```
<h2>{{ hero.name | uppercase }} Details</h2>
<div><span>id: </span>{{hero.id}}</div>
<div>
    <label>name:
      <input [(ngModel)]="hero.name" placeholder="name">
    </label>
</div>
```

src/app/app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms'; // <-- NgModel lives here

import { AppComponent } from './app.component';
import { HeroesComponent } from './heroes/heroes.component';

@NgModule({
  declarations: [
```

```
    AppComponent,
    HeroesComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

src/app/app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Tour of Heroes';
}
```

src/app/app.component.html

```
<h1>{{title}}</h1>
<app-heroes></app-heroes>
```

src/app/hero.ts

```
export class Hero {
  id: number;
  name: string;
}
```

Récapitulons! Nous avons jusqu'à maintenant:

- créé la structure d'application initiale à l'aide de l'interface angular CLI .
- appris que les composants angular affichent des données.
- utilisé les doubles accolades d'interpolation pour afficher le titre de l'application.
- utilisé l'interface de ligne de commande angular CLI pour créer un second composant HeroesComponent.
- affiché le HeroesComponent en l'ajoutant au shell AppComponent.
- appliqué la majusculePipe UppercasePipe pour mettre en forme le nom.
- utilisé la liaison de données bidirectionnelle avec la directive ngModel.
- appris à propos de l'AppModule.
- importé FormsModule dans AppModule pour qu'Angular reconnaisse et applique la directive ngModel.
- appris l'importance de déclarer les composants dans AppModule.
- apprécié le fait que l'interface CLI l'ait déclaré pour nous.

Comment Afficher Une Liste De Héros?

Dans cette section, nous allons développer l'application Tour of Heroes pour afficher une liste de héros et permettre aux utilisateurs de sélectionner un héros et d'afficher les détails du héros.

Créer Des Faux Héros

Vous aurez besoin de quelques héros pour les afficher.
Finalement, vous les obtiendrez à partir d'un serveur de données distant. Pour l'instant, vous allez créer des faux héros et prétendre qu'ils viennent du serveur.

Créez un fichier appelé mock-heroes.ts dans le dossier src / app /. Définissez une constante HEROES comme un tableau de dix héros et exportez-la. Le fichier devrait ressembler à ceci.

src/app/mock-heroes.ts

```
import { Hero } from './hero';

export const HEROES: Hero[] = [
  { id: 11, name: 'Mr. Nice' },
  { id: 12, name: 'Narco' },
  { id: 13, name: 'Bombasto' },
  { id: 14, name: 'Celeritas' },
  { id: 15, name: 'Magneta' },
  { id: 16, name: 'RubberMan' },
  { id: 17, name: 'Dynama' },
  { id: 18, name: 'Dr IQ' },
  { id: 19, name: 'Magma' },
  { id: 20, name: 'Tornado' }
];
```

Afficher Les Héros

Vous êtes sur le point d'afficher la liste des héros en haut de la composante Héros HeroesComponent.

Ouvrez le fichier de classe HeroesComponent et importez les faux HEROES.

src / app / heroes / heroes.component.ts (import HEROES)

```
import { HEROES } from '../mock-heroes';
```

Ajouter une propriété heroes à la classe qui expose ces héros pour la liaison.

```
heroes = HEROES;
```

Liste Des Héros Avec **ngFor*

Ouvrez le fichier de modèle HeroesComponent et apportez les modifications suivantes:

- Ajouter un <h2> en haut,
- juste au dessous, ajoutez une liste HTML non ordonnée (<ul>)
- Insérez un <li> dans le <ul> qui affiche les propriétés d'un héro.
- Parsemez quelques classes CSS pour le style (vous ajouterez les styles CSS prochainement).

Il Faut que ça ressemble à ceci:

heroes.component.html (modèle des héros)

```
<h2>My Heroes</h2>
<ul class="heroes">
  <li>
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>
```

Maintenant, changez le <li> en ceci:

```
<li *ngFor="let hero of heroes">
<li *ngFor="let hero of heroes">
```

Le *ngFor dans la directive du répéteur angulaire *angular's repeater*. Il répète l'élément hôte pour chaque élément d'une liste.

Dans cet exemple

- est l'élément hôte
- heroes est la liste de la classe HeroesComponent.
- hero détient l'objet héros actuel pour chaque itération à travers la liste.

Ne jamais oublier l'astérisque (*) devant ngFor. C'est une partie critique de la syntaxe.

Une fois le navigateur actualisé, la liste des héros apparaît.

Le style Des Héros

La liste des héros devrait être attrayante et devrait répondre visuellement lorsque les utilisateurs survolent et choisissent un héros de la liste. Dans le premier didacticiel, nous avons définis les styles de base pour l'ensemble de l'application dans styles.css. Cette feuille de style n'a pas inclus de styles pour cette liste de héros! Vous pouvez ajouter plus de styles à styles.css et continuer à développer cette feuille de style lorsque vous ajoutez des composants.

Vous préférerez peut-être définir des styles privés pour un composant spécifique et conserver tout ce dont un composant a besoin —le code, le code HTML et le CSS— au même endroit. Cette approche facilite la réutilisation du composant ailleurs et permet d'obtenir l'apparence souhaitée du composant, même si les styles globaux sont différents.

Vous définissez les styles privés soit en ligne dans le tableau @Component.styles, soit en tant que fichier (s) de feuille de style identifié (s) dans le tableau @Component.styleUrls.

Quand la CLI a généré le composant HeroesComponent, elle a créé une feuille de style heroes.component.css vide pour le composant HeroesComponent et l'a pointée dans @Component.styleUrls comme ceci.

rc/app/heroes/heroes.component.ts (@Component)

```
@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
})
```

Ouvrez le fichier heroes.component.css et collez les styles CSS privés pour le composant HeroesComponent. Vous les trouverez dans la revue finale du code au bas de ce guide.

Les styles et les feuilles de style identifiés dans les métadonnées @Component sont limités à ce composant spécifique. Les styles heroes.component.css s'appliquent uniquement à HeroesComponent et n'affectent pas le code HTML externe ou le code HTML d'un autre composant.

Master/Detail

Lorsque l'utilisateur clique sur un héros dans la liste principale, le composant doit afficher les détails du héros sélectionné en bas de la page.

Dans cette section, vous allez écouter l'événement de clic sur l'élément héros et mettre à jour le détail du héros.

Ajouter Une liaison D'Événement De Clic

Ajoutez un événement de clic lié à <li> comme ceci:

heroes.component.html (extrait de modèle)

```
<li *ngFor="let hero of heroes" (click)="onSelect(hero)">
```

Ceci est un exemple de la syntaxe de liaison d'événements d'Angular.

Les parenthèses autour de click indiquent à Angular d'écouter l'événement click de l'élément <li>. Lorsque l'utilisateur clique sur <li>, Angular exécute l'expression onSelect (hero).

onSelect () est une méthode HeroesComponent que vous êtes sur le point d'écrire. Angular l'appelle avec l'objet héros affiché dans le <li> cliqué, le même héro défini précédemment dans l'expression *ngFor.

Ajouter Le Gestionnaire D'Événements Click

Renommez la propriété hero du composant en selectedHero mais ne l'attribuez pas. Il n'y a pas de héro sélectionné lorsque l'application démarre.

Ajoutez la méthode onSelect () suivante, qui affecte le héros cliqué du modèle à l'élément selectedHero du composant.

src/app/heroes/heroes.component.ts (onSelect)

```
selectedHero: Hero;

onSelect(hero: Hero): void {
  this.selectedHero = hero;
}
```

La Mise À Jours Du Modèle De Détails

Le modèle fait toujours référence à l'ancienne propriété hero du composant qui n'existe plus. Renommez hero en selectedHero.

heroes.component.html (selected hero details)

```
<h2>{{ selectedHero.name | uppercase }} Details</h2>
<div><span>id: </span>{{selectedHero.id}}</div>
<div>
  <label>name:
    <input [(ngModel)]="selectedHero.name" placeholder="name">
  </label>
</div>
```

Masquer Les Détails Vides Avec *ngIf

Après l'actualisation du navigateur, l'application est interrompue.

Ouvrez les outils de développement du navigateur et recherchez dans la console un message d'erreur comme celui-ci:

```
HeroesComponent.html:3 ERROR TypeError: Cannot read property 'name' of undefined
```

Maintenant, cliquez sur l'un des éléments de la liste. L'application semble fonctionner à nouveau. Les héros apparaissent dans une liste et les détails sur le héros cliqué apparaissent au bas de la page.

Qu'Est-Ce Qui S'Est Passé?

Lorsque l'application démarre, le fichier selectedHero n'est pas défini undefined par la conception.

Les expressions de liaison dans le modèle qui font référence aux propriétés de selectedHero — expressions telles que {{selectedHero.name}} — doivent échouer car aucun héros n'est sélectionné.

Le Correctif

Le composant doit uniquement afficher les détails du héros sélectionné si le paramètre selectedHero existe.

Enveloppez le code HTML du héros dans un . Ajoutez la directive *ngIf d'Angular à et réglez-la sur selectedHero.

Ne jamais oublier l'astérisque (*) devant ngIf. C'est une partie critique de la syntaxe.

src/app/heroes/heroes.component.html (*ngIf)

```
<div *ngIf="selectedHero">

  <h2>{{ selectedHero.name | uppercase }} Details</h2>
  <div><span>id: </span>{{selectedHero.id}}</div>
  <div>
    <label>name:
      <input [(ngModel)]="selectedHero.name" placeholder="name">
    </label>
  </div>

</div>
```

Après l'actualisation du navigateur, la liste des noms réapparaît. La zone de détails est vide. Cliquez sur un héros et ses détails apparaissent.

Pourquoi Ça A Marché?

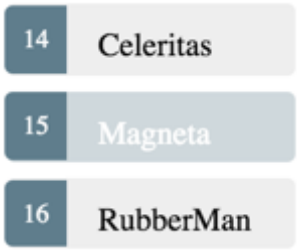
Lorsque selectedHero est indéfini, le ngIf supprime le détail du héros du DOM. Il n'y a aucune liaison selectedHero pour laquelle s'inquiéter.

Lorsque l'utilisateur choisit un héros, selectedHero prend une valeur et ngIf place le héros dans le DOM.

Style Du Héros Sélectionné

Il est difficile d'identifier le héros sélectionné dans la liste lorsque tous les éléments <li> se ressemblent.

Si l'utilisateur clique sur « Magneta », ce héro devrait afficher une couleur d'arrière-plan distincte mais subtile comme celle-ci:



Cette coloration de l'héros sélectionné est le travail de la classe CSS .selected dans les styles que vous avez ajoutés précédemment. Vous devez simplement appliquer la classe .selected à <li> lorsque l'utilisateur clique dessus.

La liaison de classe Angular facilite l'ajout et la suppression conditionnelles d'une classe CSS. Ajoutez juste [class.some-css-class] = « some-condition » à l'élément que vous voulez styler.

Ajoutez la liaison [class.selected] suivante à <li> dans le modèle HeroesComponent:

heroes.component.html (toggle the 'selected' CSS class)

```
[class.selected]="hero === selectedHero"
```

Lorsque le héros de la ligne en cours est le même que le Héros sélectionné, Angular ajoute la classe CSS sélectionnée. Lorsque les deux héros sont différents, Angular supprime la classe.

Le <li> fini ressemble à ceci:

heroes.component.html (list item hero)

```
<li *ngFor="let hero of heroes"
  [class.selected]="hero === selectedHero"
  (click)="onSelect(hero)">
  <span class="badge">{{hero.id}}</span> {{hero.name}}
```

```
</li>
```

La Revue Du Code Final

Votre application devrait ressembler à cet exemple direct / téléchargement de l'exemple.

Voici les fichiers des codes discutés sur cette page, y compris les styles HeroesComponent.

src/app/heroes/heroes.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Hero } from '../hero';
import { HEROES } from '../mock-heroes';

@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
})
export class HeroesComponent implements OnInit {

  heroes = HEROES;

  selectedHero: Hero;


  constructor() { }

  ngOnInit() {
  }

  onSelect(hero: Hero): void {
    this.selectedHero = hero;
  }
}
```

src/app/heroes/heroes.component.html

```
<h2>My Heroes</h2>
<ul class="heroes">
  <li *ngFor="let hero of heroes"
    [class.selected]="hero === selectedHero"
    (click)="onSelect(hero)">
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>

<div *ngIf="selectedHero">

  <h2>{{ selectedHero.name | uppercase }} Details</h2>
  <div><span>id: </span>{{selectedHero.id}}</div>
  <div>
    <label>name:
      <input [(ngModel)]="selectedHero.name" placeholder="name">
    </label>
  </div>

</div>
```

src/app/heroes/heroes.component.css

```
/* HeroesComponent's private CSS styles */
.selected {
  background-color: #CFD8DC !important;
  color: white;
}
.heroes {
  margin: 0 0 2em 0;
```

```
  list-style-type: none;
  padding: 0;
  width: 15em;
}
.heroes li {
  cursor: pointer;
  position: relative;
  left: 0;
  background-color: #EEE;
  margin: .5em;
  padding: .3em 0;
  height: 1.6em;
  border-radius: 4px;
}
.heroes li.selected:hover {
  background-color: #BBD8DC !important;
  color: white;
}
.heroes li:hover {
  color: #607D8B;
  background-color: #DDD;
  left: .1em;
}
.heroes .text {
  position: relative;
  top: -3px;
}
.heroes .badge {
  display: inline-block;
  font-size: small;
  color: white;
  padding: 0.8em 0.7em 0 0.7em;
  background-color: #607D8B;
  line-height: 1em;
  position: relative;
  left: -1px;
  top: -4px;
  height: 1.8em;
  margin-right: .8em;
  border-radius: 4px 0 0 4px;
}
```

Récapitulons! Nous avons jusqu'à maintenant:

- créé la structure d'application initiale à l'aide de l'interface angular CLI .
- appris que les composants angular affichent des données.
- utilisé les doubles accolades d'interpolation pour afficher le titre de l'application.
- utilisé l'interface de ligne de commande angular CLI pour créer un second composant `HeroesComponent`.
- affiché le `HeroesComponent` en l'ajoutant au shell `AppComponent`.
- appliqué la majusculePipe `UppercasePipe` pour mettre en forme le nom.
- utilisé la liaison de données bidirectionnelle avec la directive `ngModel`.
- appris à propos de l'`AppModule`.
- importé `FormsModule` dans `AppModule` pour qu'Angular reconnaisse et applique la directive `ngModel`.
- appris l'importance de déclarer les composants dans `AppModule`.
- apprécié le fait que l'interface CLI l'ait déclaré pour nous.
- vu comment l'application Tour of Heroes affiche une liste de héros dans une vue Maître / Détail.
- vu comment l'utilisateur peut sélectionner un héros et voir les détails de ce héros.
- utilisé `*ngFor` pour afficher une liste.
- utilisé `*ngFor` pour inclure ou exclure un bloc HTML de manière conditionnelle.
- pu basculer une classe de style CSS avec une liaison de `class`.