Le Composant Hero/Detail

Pour l'instant, *HeroesComponent* affiche à la fois la liste des héros et les détails du héros sélectionné.

Créer Le Composant *HeroDetailComponent*

Utilisez Angular CLI pour générer un nouveau composant nommé hero-detail.

```
ng generate component hero-detail
```

La commande échafaude les fichiers *HeroDetailComponent* et déclare le composant dans *AppModule*.

Ecrire Le Modèle

Coupez le code HTML pour le détail du héros à partir du bas du modèle *HeroesComponent* et collez-le sur le passe-partout généré dans le modèle *HeroDetailComponent*.

Le HTML collé fait référence à un *SelectedHero*. Le nouveau composant *HeroDetailComponent* peut présenter n'importe quel héros, pas seulement un héros sélectionné. Remplacez donc « *selectedHero* » par « *hero* » partout dans le modèle.

Lorsque vous aurez terminé, le modèle *HeroDetailComponent* devrait ressembler à ceci:

src/app/hero-detail/hero-detail.component.html

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

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

</div>
```

Ajoutez La Propriété @Input() Du Héro

Le modèle du composant *HeroDetailComponent* se lie à la propriété hero du composant qui est de type Hero.
Ouvrez le fichier de classe du composant *HeroDetailComponent* et importez le symbole *Hero*.

src/app/hero-detail/hero-detail.component.ts (import Hero)

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

La propriété hero doit être une propriété Input, annotée avec le décorateur *@Input()*, car le composant *HeroesComponent* externe se liera à elle comme ceci.

```
<app-hero-detail [hero]="selectedHero"></app-hero-detail>
```

Modifier l'instruction d'importation @angular/core pour inclure le symbole d'entrée.
src/app/hero-detail/hero-detail.component.ts (import Input)

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

Ajoutez la propriété *hero*, précédée du décorateur *@Input()*.

```
@Input() hero: Hero;
```

C'est le seul changement que vous devriez apporter à la classe *HeroDetailComponent*. Il n'y a plus de propriétés. Il n'y a pas de logique de présentation. Ce composant reçoit simplement un objet hero à travers sa propriété *hero* et l'affiche.

Montrer Le Composant *HeroDetailComponent*

HeroesComponent reste toujours une vue maître/détail. Il affichait auparavant les détails du héros avant de couper cette partie du modèle. Maintenant, il va déléguer cet affichage à *HeroDetailComponent*.
Les deux composants auront une relation parent/enfant. Le parent *HeroesComponent* contrôlera l'enfant *HeroDetailComponent* en lui envoyant un nouveau héros à afficher chaque fois que l'utilisateur sélectionne un héros dans la liste.
Vous ne modifierez pas la classe *HeroesComponent* mais vous modifierez son modèle.

Mettre à Jour Le Modèle Du Composant Héros

Le sélecteur *HeroDetailComponent* est '*app-hero-detail*'. Ajoutez un élément *<app-hero-detail>* au bas du modèle *HeroesComponent*, là où se trouvait la vue détaillée des héros.

Liez le *HeroesComponent.selectedHero* à la propriété *hero* de l'élément comme ceci.

```
<app-hero-detail [hero]="selectedHero"></app-hero-detail>
```

[hero] = « selectedHero« est **une liaison de propriété Angular**.

Il s'agit d'une liaison de données à sens unique entre la propriété *selectedHero* du composant *HeroesComponent* et la propriété hero de l'élément cible, qui correspond à la propriété hero du composant *HeroDetailComponent*.

Maintenant, lorsque l'utilisateur clique sur un héros dans la liste, le *selectedHero* change. Lorsque le paramètre *selectedHero* est modifié, la liaison de propriété met à jour hero et le composant *HeroDetailComponent* affiche le nouveau héro.

Le modèle du composant *HeroesComponent* révisé devrait ressembler à ceci:

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>

<app-hero-detail [hero]="selectedHero"></app-hero-detail>
```

Le navigateur se rafraîchit et l'application recommence à fonctionner comme auparavant.

Qu'Est Ce Qui A Changé?

Comme précédemment (section Comment Afficher Une Liste De Héros?) , chaque fois qu'un utilisateur clique sur un nom de héro, le détail du héro apparaît sous la liste des héros. Maintenant, *HeroDetailComponent* présente ces détails au lieu de *HeroesComponent*.

Refactoriser le composant *HeroesComponent* original en deux composants apporte des avantages, à la fois maintenant et dans le futur:

- Vous avez simplifié *HeroesComponent* en réduisant ses responsabilités.
- Vous pouvez faire évoluer *HeroDetailComponent* dans un éditeur de héros riche sans toucher le parent *HeroesComponent*.

- Vous pouvez faire évoluer le composant *HeroesComponent* sans toucher à la vue détaillée des héros.
- Vous pouvez réutiliser le *HeroDetailComponent* dans le modèle d'un futur composant.

Le Code Final

Voici les fichiers du code discutés sur cette page et votre application devrait ressembler à cet ***exemple en direct***. ou si vous voulez ***télécharger cet exemple***.

src/app/hero-detail/hero-detail.component.ts

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

@Component({
  selector: 'app-hero-detail',
  templateUrl: './hero-detail.component.html',
  styleUrls: ['./hero-detail.component.css']
})
export class HeroDetailComponent implements OnInit {
  @Input() hero: Hero;

  constructor() { }

  ngOnInit() {
  }

}
```

src/app/hero-detail/hero-detail.component.html

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

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

</div>
```

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>

<app-hero-detail [hero]="selectedHero"></app-hero-detail>
```

Récapitulons! Dans cette section nous avons:

- créé un composant Hero*DetailComponent* distinct et réutilisable.
- utilisé une liaison de propriété pour attribuer le contrôle *HeroesComponent* parent à l'objet enfant *HeroDetailComponent*.
- utilisé le ***décorateur @Input*** pour rendre la propriété hero disponible pour la liaison par le composant *HeroesComponent* externe.

Services

Jusqu'ici le composant *HeroesComponent* de la Tour of Heroes ne reçoit et n'affiche que de fausses données.

Après le refactoring dans ce tutoriel, *HeroesComponent* sera allégé et axé sur la prise en charge de la vue. Il sera également plus facile de tester un appareil avec un service simulé.

Pourquoi les services

Les composants ne doivent pas extraire ou enregistrer des données directement et ils ne devraient certainement pas présenter consciemment de fausses données. Ils devraient se concentrer sur la présentation des données et déléguer l'accès aux données à un service.

Dans cette section, nous allons créer un *HeroService* que toutes les classes d'application peuvent utiliser pour obtenir des héros. Au lieu de créer ce service avec *new*, nous comptons sur ***dependency injection*** (l'injection de dépendance) Angular pour l'injecter dans le constructeur *HeroesComponent*.

Les services sont un excellent moyen de partager les informations entre les classes qui ne se connaissent pas. Vous allez créer un *MessageService* et l'injecter à deux endroits:

1. dans *HeroService* qui utilise le service pour envoyer un message.
2. dans *MessagesComponent* qui affiche ce message.

Créer le HeroService

À l'aide de l'interface de ligne de commande Angular CLI, créez un service appelé *hero*.

```
ng generate service hero
```

La commande génère une classe *HeroService* squelette dans *src / app / hero.service.ts* La classe *HeroService* doit ressembler à l'exemple suivant.

src/app/hero.service.ts (new service)

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

@Injectable({
  providedIn: 'root',
})
export class HeroService {

  constructor() { }

}
```

Services @Injectable ()

Notez que le nouveau service importe le symbole Angular *Injectable* et annote la classe avec le décorateur *@Injectable ()*. Cela marque la classe comme une personne qui participe au système d'injection de dépendance. La classe *HeroService* va fournir un service injectable, et elle peut aussi avoir ses propres dépendances injectées. Il n'a pas encore de dépendances, mais ça ***le sera bientôt***.

Le décorateur *@Injectable ()* accepte un objet de métadonnées pour le service, de la même manière que le décorateur *@Component ()* pour vos classes de composants

Obtenir les données de hero

HeroService peut obtenir des données de héros de n'importe où — un service Web, un stockage local ou une source de données fictive.

Supprimer l'accès aux données des composants signifie que vous pouvez changer d'avis sur l'implémentation à tout moment, sans toucher à aucun composant.

La mise en œuvre dans cet article continuera à livrer des héros fictifs.

Importez le héros *Hero* et les *héros*.

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

Ajoutez une méthode *getHeroes* pour renvoyer les faux héros.

```
getHeroes(): Hero[] {
  return HEROES;
}
```

Fournir le HeroService

Vous devez rendre le *HeroService* disponible pour le système d'injection de dépendances avant que Angular puisse l'injecter dans *HeroesComponent*, comme vous le ferez ci-dessous. Vous faites cela en enregistrant un fournisseur. Un fournisseur « *provider*« est quelque chose qui peut créer ou fournir un service; dans ce cas, il instancie la classe *HeroService* pour fournir le service.

Maintenant, vous devez vous assurer que *HeroService* est enregistré en tant que fournisseur de ce service. Vous l'enregistrez avec un injecteur, qui est l'objet responsable du choix et de l'injection du fournisseur là où cela est nécessaire.

Par défaut, la commande CLI angular *ng generate service* enregistre un fournisseur auprès de l'injecteur racine pour votre service en incluant les métadonnées du fournisseur dans le décorateur *@Injectable*.

Si vous regardez l'instruction *@Injectable()* juste avant la définition de la classe *HeroService*, vous pouvez voir que la valeur de la métadonnée *providedIn* est 'root':

```
@Injectable({
  providedIn: 'root',
})
```

Lorsque vous fournissez le service au niveau racine, Angular crée une seule instance partagée de *HeroService* et l'injecte dans n'importe quelle classe qui le demande. L'enregistrement du fournisseur dans les métadonnées *@Injectable* permet également à Angular d'optimiser une application en supprimant le service s'il s'avère qu'il ne doit pas être utilisé après tout.

Si nécessaire, vous pouvez enregistrer des fournisseurs à différents niveaux: dans *HeroesComponent*, dans *AppComponent*, dans *AppModule*. Par exemple, vous pourriez avoir demandé à la CLI de fournir automatiquement le service au niveau du module en ajoutant *—module = app.*

```
ng generate service hero --module=app
```

Pour en savoir plus sur les fournisseurs et les injecteurs, consultez le guide d'injection de dépendances (***Dependency Injection guide***).

Le *HeroService* est maintenant prêt à se connecter au composant *HeroesComponent*.

Ceci est un exemple de code interimaire qui vous permettra de fournir et d'utiliser le *HeroService*. À ce stade, le code différera de *HeroService* dans la « revue du code final ».

Mettre à jour le composant Héros *HeroesComponent*

Ouvrez le fichier de classe *HeroesComponent*.

Supprimez l'importation *HEROES*, car vous n'en aurez plus besoin. Importer le *HeroService* à la place.

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

```
import { HeroService } from '../hero.service';
```

Remplacez la définition de la propriété *heroes* par une simple déclaration.

```
heroes: Hero[];
```

Injecter *HeroService*

Ajoutez un paramètre *heroService* privé de type HeroService au constructeur.

```
constructor(private heroService: HeroService) { }
```

Le paramètre définit simultanément une propriété *heroService* privée et l'identifie comme site d'injection *HeroService*.

Lorsque Angular crée un *HeroesComponent*, le système Injection de dépendances définit le paramètre *heroService* sur l'instance singleton de *HeroService*.

Ajouter *getHeroes()*

Créez une fonction pour récupérer les héros du service.

```
getHeroes(): void {
  this.heroes = this.heroService.getHeroes();
}
```

Appelez-la dans *ngOnInit*

Bien que vous puissiez appeler *getHeroes()* dans le constructeur, ce n'est pas la meilleure pratique.

Réservez le constructeur pour une initialisation simple telle que les paramètres du constructeur de câblage aux propriétés. Le constructeur ne devrait rien faire. Il ne devrait certainement pas appeler une fonction qui envoie des requêtes HTTP à un serveur distant comme le ferait un vrai service de données.

Au lieu de cela, appelez *getHeroes()* à l'intérieur du hook de cycle de vie *ngOnInit* et laissez Angular appeler *ngOnInit* à un moment approprié après la construction d'une instance *HeroesComponent*.

```
ngOnInit() {
  this.getHeroes();
}
```

Une fois le navigateur actualisé, l'application doit être exécutée comme précédemment, en affichant une liste de héros et une vue détaillée des héros lorsque vous cliquez sur un nom de héros.

Données observables

La méthode *HeroService.getHeroes()* a une signature synchrone, ce qui implique que *HeroService* peut récupérer les héros de manière synchrone. Le composant *HeroesComponent* consomme le résultat *getHeroes()* comme si les héros pouvaient être récupérés de manière synchrone.

```
this.heroes = this.heroService.getHeroes();
```

Cela ne fonctionnera pas dans une vraie application. Vous vous en sortez maintenant parce que le service renvoie actuellement des faux héros. Mais bientôt l'application va chercher des héros à partir d'un serveur distant, ce qui est une opération intrinsèquement asynchrone.

HeroService doit attendre que le serveur réponde, *getHeroes()* ne peut pas retourner immédiatement avec des données de héros, et le navigateur ne se bloquera pas pendant que le service attend.

HeroService.getHeroes() doit avoir une signature asynchrone quelconque.

Cela peut prendre un rappel. Il pourrait retourner une promesse. Il pourrait retourner un Observable.

Dans ce tutoriel, *HeroService.getHeroes()* retournera un Observable en partie parce qu'il utilisera éventuellement la méthode Angular *HttpClient.get* pour récupérer les héros et *HttpClient.get()* renvoie un Observable.

HeroService observable

Observable est l'une des classes clés de la bibliothèque *RxJS*.

Dans un dprochain tutoriel sur HTTP, vous apprendrez que les méthodes *HttpClient* d'Angular renvoient des Observables *RxJS*. Dans cette section, vous allez simuler l'obtention de données à partir du serveur avec la fonction *RxJS of()*.

Ouvrez le fichier *HeroService* et importez les symboles Observable et de RxJS.

src/app/hero.service.ts (Observable imports)

```
import { Observable, of } from 'rxjs';
```

Remplacez la méthode getHeroes par celle-ci.

```
getHeroes(): Observable<Hero[]> {
  return of(HEROES);
}
```

of(HEROES) renvoie un *Observable<Hero[]>* qui émet une seule valeur, le tableau des faux héros.

Dans le ***tutoriel HTTP***, vous appelez *HttpClient.get <Hero[]>()* qui renvoie également un *Observable<Hero[]>* qui émet une seule valeur, un tableau de héros provenant du corps de la réponse HTTP.

S'abonnez à HeroesComponent

La méthode *HeroService.getHeroes* utilisée pour renvoyer un *Hero[]*. Maintenant, il retourne un *Observable<Hero[]>*.

Vous devrez vous adapter à cette différence dans *HeroesComponent*.

Trouvez la méthode g*etHeroes* et remplacez-la par le code suivant (affiché côte à côte avec la version précédente pour comparaison)

heroes.component.ts (Observable)

```
getHeroes(): void {
  this.heroService.getHeroes()
      .subscribe(heroes => this.heroes = heroes);
}
```

heroes.component.ts (Original)

```
getHeroes(): void {
  this.heroes = this.heroService.getHeroes();
}
```

Observable.subscribe() est la différence critique.

La version précédente assigne un tableau de héros à la propriété *heroes* du composant. L'affectation se produit de manière synchrone, comme si le serveur pouvait renvoyer des héros instantanément ou que le navigateur pouvait bloquer l'interface utilisateur alors qu'il attendait la réponse du serveur.

Cela ne fonctionnera pas lorsque *HeroService* est en train de faire des requêtes sur un serveur distant.

La nouvelle version attend que l'*Observable* émette le tableau des héros, ce qui pourrait arriver maintenant ou dans quelques minutes. *Subscribe* passe ensuite le tableau émis au callback, qui définit la propriété *heroes* du composant.

Cette approche asynchrone fonctionne lorsque *HeroService* demande des héros au serveur.

Afficher les messages

Dans cette section, vous allez

- ajoutez un composant *MessagesComponent* qui affiche les messages de l'application en bas de l'écran.
- créer un *MessageService* injectable à l'échelle de l'application pour l'envoi des messages à afficher.
- injecter *MessageService* dans le *HeroService*
- afficher un message lorsque *HeroService* récupère les héros avec succès.

Créer un composant de message

Utilisez l'interface de ligne de commande CLI pour créer le composant *MessagesComponent*.

```
ng generate component messages
```

L'interface de ligne de commande CLI crée les fichiers de composants dans le dossier src/app/messages et déclare *MessagesComponent* dans *AppModule*.

Modifiez le modèle *AppComponent* pour afficher le *MessageComponent* généré
/src/app/app.component.htm

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

Vous devriez voir le paragraphe par défaut du composant Messages en bas de la page.
Créer le *MessageService*
Utilisez l'interface de ligne de commande pour créer le *MessageService* dans *src/app*.

```
ng generate service message
```

Ouvrez *MessageService* et remplacez son contenu par ce qui suit.
/src/app/message.service.ts

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

@Injectable({
  providedIn: 'root',
})
export class MessageService {
  messages: string[] = [];

  add(message: string) {
    this.messages.push(message);
  }

  clear() {
    this.messages = [];
  }
```

```
}
```

Le service expose son cache de messages et deux méthodes: une pour ajouter *add()* un message au cache et une autre pour *clear()* le cache.

Injecter dans le *HeroService*

Ré-ouvrez le *HeroService* et importez le *MessageService*.

/src/app/hero.service.ts (import MessageService)

```
import { MessageService } from './message.service';
```

Modifiez le constructeur avec un paramètre qui déclare une propriété messageService privée. Angular va injecter le MessageService singleton dans cette propriété quand il crée le HeroService.

```
constructor(private messageService: MessageService) { }
```

Ceci est un scénario typique de « service en service »: vous injectez le *MessageService* dans le *HeroService* qui est injecté dans le composant *HeroesComponent*.

Envoyer un message depuis HeroService

Modifiez la méthode *getHeroes* pour envoyer un message lorsque les héros sont récupérés.

```
getHeroes(): Observable<Hero[]> {
  // TODO: send the message _after_ fetching the heroes
  this.messageService.add('HeroService: fetched heroes');
  return of(HEROES);
}
```

Afficher le message depuis *HeroService*
Le composant *MessagesComponent* doit afficher tous les messages, y compris le message envoyé par *HeroService* lorsqu'il récupère des héros.

Ouvrez *MessagesComponent* et importez le *MessageService*.
/src/app/messages/messages.component.ts (import MessageService)

```
import { MessageService } from '../message.service';
```

Modifiez le constructeur avec un paramètre qui déclare une propriété publique *messageService*. Angular va injecter le *MessageService* singleton dans cette propriété lorsqu'il crée le composant *MessagesComponent*.

```
constructor(public messageService: MessageService) {}
```

La propriété *messageService* doit être publique car vous êtes sur le point de la lier dans le modèle.

Angular se lie uniquement aux propriétés des composants publics.

Lier au *MessageService*

Remplacez le modèle *MessagesComponent* généré par CLI par ce qui suit.

src/app/messages/messages.component.html

```
<div *ngIf="messageService.messages.length">

  <h2>Messages</h2>
  <button class="clear"
          (click)="messageService.clear()">clear</button>
  <div *ngFor='let message of messageService.messages'> {{message}} </div>
```

```
</div>
```

Ce modèle se lie directement au *messageService* du composant.

- **ngIf* affiche uniquement la zone des messages s'il y a des messages à afficher.
- **ngFor* présente la liste des messages dans les éléments répétés * *.
- Une liaison d'événement angular lie l'événement click du bouton à *MessageService.clear()*.

Les messages apparaîtront mieux lorsque vous ajouterez les styles CSS privés à *messages.component.css* comme indiqué dans l'un des onglets « Revue finale du code » ci-dessous.

Le navigateur se rafraîchit et la page affiche la liste des héros. Faites défiler vers le bas pour voir le message du *HeroService* dans la zone de message. Cliquez sur le bouton « Clear » et la zone de message disparaît.

Revue Finale Du code

Voici les fichiers des codes discutés sur cette page, votre application devrait ressembler à cet ***exemple en direct*** / ***téléchargement de l'exemple***.

src/app/hero.service.ts

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

import { Observable, of } from 'rxjs';

import { Hero } from './hero';
import { HEROES } from './mock-heroes';
import { MessageService } from './message.service';

@Injectable({
  providedIn: 'root',
})
export class HeroService {

  constructor(private messageService: MessageService) { }

  getHeroes(): Observable<Hero[]> {
    // TODO: send the message _after_ fetching the heroes
    this.messageService.add('HeroService: fetched heroes');
    return of(HEROES);
  }
}
```

src/app/message.service.ts

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

@Injectable({
  providedIn: 'root',
})
export class MessageService {
  messages: string[] = [];

  add(message: string) {
    this.messages.push(message);
  }

  clear() {
    this.messages = [];
  }
}
```

src/app/heroes/heroes.component.ts

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

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

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

  selectedHero: Hero;

  heroes: Hero[];

  constructor(private heroService: HeroService) { }

  ngOnInit() {
    this.getHeroes();
  }

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

  getHeroes(): void {
    this.heroService.getHeroes()
        .subscribe(heroes => this.heroes = heroes);
  }
}
```

src/app/messages/messages.component.ts

```
import { Component, OnInit } from '@angular/core';
import { MessageService } from '../message.service';

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

  constructor(public messageService: MessageService) {}

  ngOnInit() {
  }

}
```

src/app/messages/messages.component.html

```
<div *ngIf="messageService.messages.length">

  <h2>Messages</h2>
  <button class="clear"
          (click)="messageService.clear()">clear</button>
  <div *ngFor='let message of messageService.messages'> {{message}} </div>

</div>
```

src/app/messages/messages.component.css

```
/* MessagesComponent's private CSS styles */
h2 {
  color: red;
  font-family: Arial, Helvetica, sans-serif;
  font-weight: lighter;
}
body {
  margin: 2em;
```

```
}
body, input[text], button {
  color: crimson;
  font-family: Cambria, Georgia;
}

button.clear {
  font-family: Arial;
  background-color: #eee;
  border: none;
  padding: 5px 10px;
  border-radius: 4px;
  cursor: pointer;
  cursor: hand;
}
button:hover {
  background-color: #cfd8dc;
}
button:disabled {
  background-color: #eee;
  color: #aaa;
  cursor: auto;
}
button.clear {
  color: #888;
  margin-bottom: 12px;
}
```

src/app/app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { HeroesComponent } from './heroes/heroes.component';
import { HeroDetailComponent } from './hero-detail/hero-detail.component';
import { MessagesComponent } from './messages/messages.component';

@NgModule({
  declarations: [
    AppComponent,
    HeroesComponent,
    HeroDetailComponent,
    MessagesComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [
    // no need to place any providers due to the `providedIn` flag...
  ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

src/app/app.component.html

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

Récapitulons! dans cet article:

- Vous avez remanié l'accès aux données de la classe HeroService.
- Vous avez enregistré le HeroService en tant que fournisseur de son service au niveau de la racine afin qu'il puisse être injecté n'importe où dans l'application.
- Vous avez utilisé l'Injection de dépendance angular pour l'injecter dans un composant.
- Vous avez donné à la méthode HeroService get data une signature asynchrone.

- Vous avez découvert Observable et la bibliothèque RxJS Observable.
- Vous avez utilisé RxJS de() pour retourner une observable de faux héros (Observable <Hero []>).
- Le hook de cycle de vie ngOnInit du composant appelle la méthode HeroService, pas le constructeur.
- Vous avez créé un MessageService pour une communication faiblement couplée entre les classes.
- Le HeroService injecté dans un composant est créé avec un autre service injecté, MessageService.