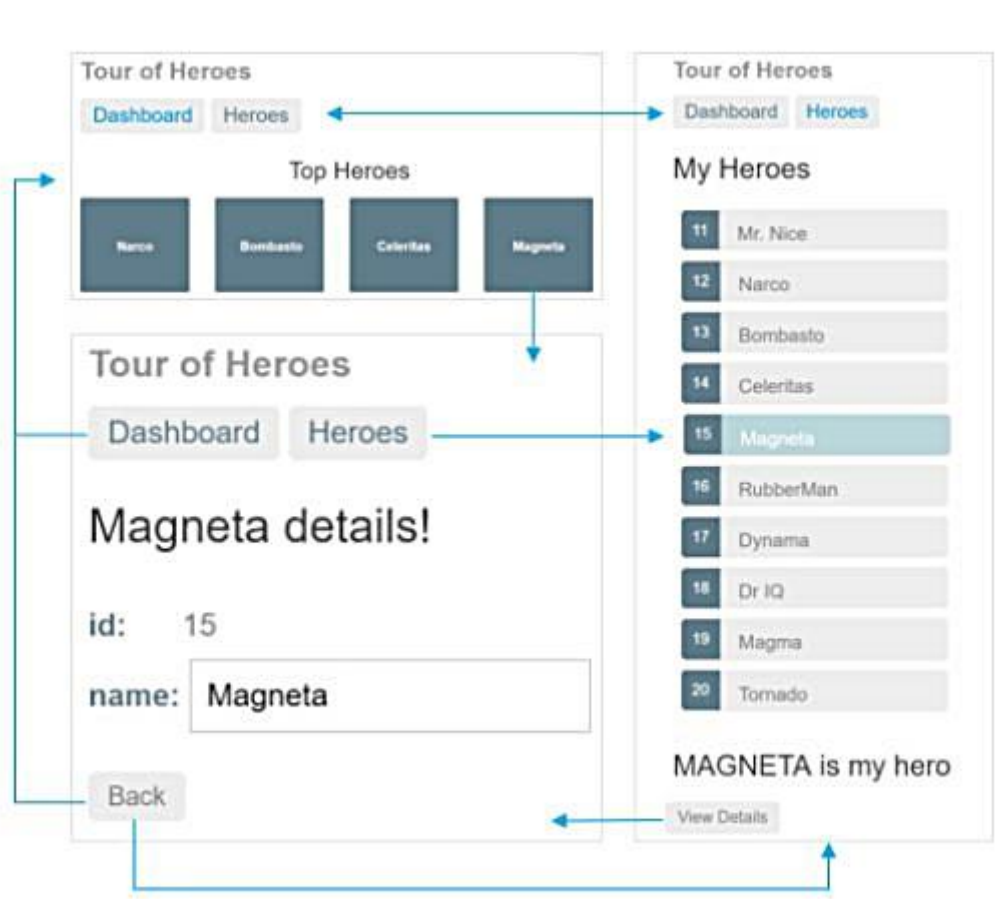
Routage: Ajouter le module *AppRoutingModule*

Il y a de nouvelles exigences pour l'application Tour of Heroes:

- Ajouter une vue du tableau de bord (Dashboard)
- Ajoutez la possibilité de naviguer entre les vues Heroes et le tableau de bord
- Lorsque les utilisateurs cliquent sur un nom de héros dans l'une des vues, accédez à une vue détaillée du héros sélectionné.
- Lorsque les utilisateurs cliquent sur un lien profond dans un e-mail, ouvrez la vue détaillée d'un héros particulier.

Lorsque vous aurez terminé, les utilisateurs pourront naviguer dans l'application comme ceci:



Une meilleure pratique angulaire consiste à charger et à configurer le routeur dans un module de niveau supérieur distinct, dédié au routage et importé par l'*AppModule* racine.

Par convention, le nom de classe du module est *AppRoutingModule* et il appartient à l'*app-routing.module.ts* dans le dossier *src/app*.

Utilisez l'interface de ligne de commande CLI pour le générer.

```
ng generate module app-routing --flat --module=app
```

–flat place le fichier dans *src/app* au lieu de son propre dossier.
–module=app demande à la CLI de l'enregistrer dans le tableau imports de l'*AppModule*
Le fichier généré ressemble à ceci:

```
src / app / app-routing.module.ts (généré)


import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
```

```
@NgModule({
  imports: [
    CommonModule
  ],
  declarations: []
})
export class AppRoutingModule { }
```

En règle générale, vous ne déclarez pas les composants dans un module de routage, vous pouvez donc supprimer le tableau *@NgModule.declarations* et supprimer également les références *CommonModule*.

Vous allez configurer le routeur avec *Routes* dans *RouterModule*, donc importez ces deux symboles depuis la bibliothèque *@angular/router*.

Ajoutez un tableau *@NgModule.exports* avec *RouterModule*. L'exportation de *RouterModule* permet d'utiliser les directives de routeur dans les composants *AppModule* qui en auront besoin.

Maintenant, *AppRoutingModule* ressemble à ceci:

```
src/app/app-routing.module.ts (v1)


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

@NgModule({
  exports: [ RouterModule ]
})
export class AppRoutingModule {}
```

Ajouter *Routes*

Routes indiquent au routeur la vue à afficher lorsqu'un utilisateur clique sur un lien ou colle une URL dans la barre d'adresse du navigateur.

Route typique a deux propriétés:

1. *path*: une chaîne qui correspond à l'URL dans la barre d'adresse du navigateur.
2. *component* que le routeur doit créer lors de la navigation vers ce *Route*.

Vous avez l'intention de naviguer vers le *HeroesComponent* lorsque l'URL est quelque chose comme *localhost:4200/heroes*.

Importez le composant *HeroesComponent* pour pouvoir le référencer dans *Route*. Définissez ensuite un tableau de routes avec un chemin unique vers ce composant.

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

const routes: Routes = [
  { path: 'heroes', component: HeroesComponent }
];
```

Une fois la configuration terminée, le routeur associera cette URL au chemin: '*heroes*' et affichera le *HeroesComponent*.

RouterModule.forRoot()

Vous devez d'abord initialiser le routeur et le démarrer en écoutant les changements d'emplacement du navigateur. Ajoutez *RouterModule* au tableau *@NgModule.imports* et configurez-le avec les routes en une étape en appelant *RouterModule.forRoot()* dans le tableau imports, comme ceci:

```
imports: [ RouterModule.forRoot(routes) ],
```

La méthode s'appelle *forRoot()* car vous configurez le routeur au niveau racine de l'application. La méthode *forRoot()* fournit les fournisseurs de services et les directives nécessaires au routage et effectue la navigation initiale en fonction de l'URL du navigateur actuel.

Ajouter *RouterOutlet*

Ouvrez le modèle AppComponent en remplaçant l'élément <*app-heroes*> par un élément <*router-outlet*>.

src/app/app.component.html (router-outlet)

```
<h1>{{title}}</h1>
<router-outlet></router-outlet>
<app-messages></app-messages>
```

Vous avez supprimé <*app-heroes*> car vous n'afficherez le composant *HeroesComponent* que lorsque l'utilisateur y accède.

Le <*routeur-outlet*> indique au routeur où afficher les vues routées.

RouterOutlet est l'une des directives de routeur qui sont devenues disponibles pour *AppComponent* car *AppModule* importe *AppRoutingModule* qui a exporté *RouterModule*.

Faites vos essais

Vous devriez toujours utiliser cette commande CLI.

```
ng serve
```

Le navigateur doit actualiser et afficher le titre de l'application, mais pas la liste des héros.

Regardez la barre d'adresse du navigateur. L'URL se termine par /. Le chemin d'accès à *HeroesComponent* est */heroes*.

Ajouter */heroes* à l'URL dans la barre d'adresse du navigateur. Vous devriez voir la vue master/détail des héros familiers.

Add a navigation link (routerLink)

Les utilisateurs ne doivent pas avoir à coller une URL du chemin dans la barre d'adresse. Ils devraient être en mesure de cliquer sur un lien pour naviguer.

Ajoutez un élément <*nav*> et, à l'intérieur, un élément d'ancrage qui, lorsqu'il est cliqué, déclenche la navigation vers *HeroesComponent*. Le modèle *AppComponent* révisé ressemble à ceci:

src/app/app.component.html (heroes RouterLink)

```
<h1>{{title}}</h1>
<nav>
  <a routerLink="/heroes">Heroes</a>
</nav>
<router-outlet></router-outlet>
<app-messages></app-messages>
```

Un attribut *routerLink* est défini sur « */heroes*« , la chaîne que le routeur correspond au chemin vers *HeroesComponent*. *RouterLink* est le sélecteur de la directive *RouterLink* qui transforme les clics utilisateur en navigations de routeur. C'est une autre directive publique dans *RouterModule*.

Le navigateur actualise et affiche le lien du titre de l'application et des héros, mais pas la liste des héros.

Cliquer sur le lien. La barre d'adresse est mise à jour pour */heroes* et la liste des héros apparaît.

Faites en sorte que ce lien et les futurs liens de navigation soient plus beaux en ajoutant des styles CSS privés à *app.component.css* comme indiqué dans la revue de code finale ci-dessous.

Ajouter une vue de tableau de bord

Le routage a plus de sens lorsqu'il y a plusieurs vues. Jusqu'à présent, il n'y a que la vue des héros.
Ajouter un composant de tableau de bord à l'aide de l'interface de ligne de commande CLI:

```
ng generate component dashboard
```

L'interface de ligne de commande CLI génère les fichiers pour *DashboardComponent* et les déclare dans *AppModule*.

Remplacez le contenu de fichier par défaut dans ces trois fichiers comme suit, puis revenez pour une petite discussion:

src/app/dashboard/dashboard.component.html

```
<h3>Top Heroes</h3>
<div class="grid grid-pad">
  <a *ngFor="let hero of heroes" class="col-1-4">
    <div class="module hero">
      <h4>{{hero.name}}</h4>
    </div>
  </a>
</div>
```

src/app/dashboard/dashboard.component.ts

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

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

  constructor(private heroService: HeroService) { }

  ngOnInit() {
    this.getHeroes();
  }

  getHeroes(): void {
    this.heroService.getHeroes()
      .subscribe(heroes => this.heroes = heroes.slice(1, 5));
  }
}
```

src/app/dashboard/dashboard.component.css

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

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

```
})
export class DashboardComponent implements OnInit {
  heroes: Hero[] = [];

  constructor(private heroService: HeroService) { }

  ngOnInit() {
    this.getHeroes();
  }

  getHeroes(): void {
    this.heroService.getHeroes()
      .subscribe(heroes => this.heroes = heroes.slice(1, 5));
  }
}
```

Le modèle présente une grille de liens de noms de héros.

- Le répéteur **ngFor* crée autant de liens que dans le tableau des héros du composant.
- Les liens sont stylisés en tant que blocs colorés par le *dashboard.component.css*.
- Les liens ne vont nulle part encore mais ils le feront sous peu.

La classe est similaire à la classe *HeroesComponent*.

- Il définit une propriété de tableau de héros.
- Le constructeur s'attend à ce que Angular injecte le *HeroService* dans une propriété *heroService* privée.
- Le hook de cycle de vie *ngOnInit()* appelle *getHeroes*.

Ce *getHeroes* réduit le nombre de héros affichés à quatre (2e, 3e, 4e et 5e).

```
getHeroes(): void {
  this.heroService.getHeroes()
    .subscribe(heroes => this.heroes = heroes.slice(1, 5));
}
```

Ajouter la route du tableau de bord

Pour naviguer vers le tableau de bord, le routeur a besoin d'un chemin approprié.

Importez le *DashboardComponent* dans *AppRoutingModule*.

```
src/app/app-routing.module.ts (import DashboardComponent)


import { DashboardComponent }   from './dashboard/dashboard.component';
```

Ajoutez un chemin au tableau *AppRoutingModule.routes* qui correspond à un chemin d'accès au composant *DashboardComponent*.

```
{ path: 'dashboard', component: DashboardComponent },
```

Ajouter une route par défaut

Lorsque l'application démarre, la barre d'adresse du navigateur pointe vers la racine du site Web. Cela ne correspond à aucun chemin existant, de sorte que le routeur ne navigue nulle part. L'espace situé sous le *<routeur-outlet>* est vide.

Pour que l'application accède automatiquement au tableau de bord, ajoutez la route suivante au tableau *AppRoutingModule.Routes*.

```
{ path: '', redirectTo: '/dashboard', pathMatch: 'full' },
```

Ce chemin redirige une URL qui correspond entièrement au chemin vide de la route dont le chemin est *'/dashboard'*.

Une fois le navigateur actualisé, le routeur charge le composant *DashboardComponent* et la barre d'adresse du navigateur affiche l'URL */dashboard*.

Ajouter un lien de tableau de bord l'interface d'utilisateur (shell)

L'utilisateur devrait pouvoir naviguer entre *DashboardComponent* et *HeroesComponent* en cliquant sur les liens dans la zone de navigation située en haut de la page.

Ajoutez un lien de navigation de tableau de bord au modèle de shell *AppComponent*, juste au-dessus du lien Héros.

src/app/app.component.html

```
<h1>{{title}}</h1>
<nav>
  <a routerLink="/dashboard">Dashboard</a>
  <a routerLink="/heroes">Heroes</a>
</nav>
<router-outlet></router-outlet>
<app-messages></app-messages>
```

Après l'actualisation du navigateur, vous pouvez naviguer librement entre les deux vues en cliquant sur les liens.

Naviguer vers les détails des héros

Le composant *HeroDetailsComponent* affiche les détails d'un héros sélectionné. Pour le moment, le composant *HeroDetailsComponent* n'est visible qu'au bas du composant *HeroesComponent*

L'utilisateur devrait pouvoir accéder à ces informations de trois façons.

1. En cliquant sur un héros dans le tableau de bord.
2. En cliquant sur un héros dans la liste des héros.
3. En collant une URL « lien profond » dans la barre d'adresse du navigateur qui identifie le héros à afficher.

Dans cette section, vous allez activer la navigation vers *HeroDetailsComponent* et la libérer du composant *HeroesComponent*.

Supprimer les détails du héros de *HeroesComponent*

Lorsque l'utilisateur clique sur un élément de héros dans *HeroesComponent*, l'application doit accéder au composant *HeroDetailComponent*, en remplaçant la vue de la liste des héros par la vue de détail des héros. La vue de la liste des héros ne devrait plus montrer les détails des héros comme c'est le cas maintenant.

Ouvrez le modèle *HeroesComponent* (*heroes/heroes.component.html*) et supprimez l'élément *<app-hero-detail>* en bas.

Cliquer sur un objet héros ne fait plus rien. Vous corrigerez cela peu de temps après avoir activé le routage vers *HeroDetailComponent*.

Ajouter un chemin de détails pour les héros

Une URL comme *~/detail/11* serait une bonne URL pour naviguer vers la vue Hero Detail du héros dont l'identifiant est 11.

Ouvrez *AppRoutingModule* et importez *HeroDetailComponent*.

src/app/app-routing.module.ts (import HeroDetailComponent)

```
import { HeroDetailComponent }  from './hero-detail/hero-detail.component';
```

Ajoutez ensuite un chemin paramétrée au tableau *AppRoutingModule.routes* qui correspond au modèle de chemin d'accès à la vue détaillée des héros.

```
{ path: 'detail/:id', component: HeroDetailComponent },
```

Le signe deux-points *(:)* dans le chemin indique que *:id* est un espace réservé pour un identifiant de héros spécifique.

À ce stade, tous les chemins d'application sont en place.

```
src/app/app-routing.module.ts (all routes)


const routes: Routes = [
  { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
  { path: 'dashboard', component: DashboardComponent },
  { path: 'detail/:id', component: HeroDetailComponent },
  { path: 'heroes', component: HeroesComponent }
];
```

Liens de héros *DashboardComponent*

Les liens de héros *DashboardComponent* ne font rien pour le moment.

Maintenant que le routeur a un chemin vers *HeroDetailComponent*, corrigez les liens du héros du tableau de bord pour naviguer via le chemin du tableau de bord paramétrée.

```
src/app/dashboard/dashboard.component.html (hero links)


<a *ngFor="let hero of heroes" class="col-1-4"
    routerLink="/detail/{{hero.id}}">
```

Vous utilisez une liaison d'interpolation angular dans le répéteur **ngFor* pour insérer le fichier *hero.id* de l'itération courante dans chaque *routerLink*.

HeroesComponent liens de héros

Les éléments de héros dans *HeroesComponent* sont des éléments *<li>* dont les événements de clic sont liés à la méthode *onSelect()* du composant.

```
src/app/heroes/heroes.component.html (liste avec onSelect)


<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>
```

Enlever le le *<li>* juste à son **ngFor*, envelopper le badge et le nom dans un élément d'ancrage (*<a>*), et ajouter un attribut *routerLink* à l'ancre qui est le même que dans le modèle de tableau de bord

```
src/app/heroes/heroes.component.html (liste avec liens)


<ul class="heroes">
  <li *ngFor="let hero of heroes">
    <a routerLink="/detail/{{hero.id}}">
      <span class="badge">{{hero.id}}</span> {{hero.name}}
    </a>
  </li>
</ul>
```

Vous devrez corriger la feuille de style privée (*heroes.component.css*) pour que la liste ressemble à ce qu'elle était avant. Les styles révisés figurent dans la revue finale du code au bas de cette section.

Supprimer le code mort (facultatif)

Alors que la classe *HeroesComponent* fonctionne toujours, la méthode *onSelect()* et la propriété *selectedHero* ne sont plus utilisées.

C'est agréable de ranger les choses et vous vous en serez reconnaissant plus tard. Voici la classe après avoir éliminé le code mort.

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

```
export class HeroesComponent implements OnInit {
  heroes: Hero[];

  constructor(private heroService: HeroService) { }

  ngOnInit() {
    this.getHeroes();
  }

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

Routable *HeroDetailComponent*

Auparavant, le parent *HeroesComponent* définissait la propriété *HeroDetailComponent.hero* et le composant *HeroDetailComponent* affichait le héros.

HeroesComponent ne le fait plus. Maintenant, le routeur crée *HeroDetailComponent* en réponse à une URL telle que *~/detail/11*.

HeroDetailComponent a besoin d'une nouvelle façon d'obtenir le héros à afficher.

- Obtenez le chemin qui l'a créé
- Extraire l'identifiant *id* du chemin
- Acquérir le héros avec cet *id* du serveur via le *HeroService*

Ajouter les importations suivantes:

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

```
import { ActivatedRoute } from '@angular/router';
import { Location } from '@angular/common';

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

Injectez les services *ActivatedRoute*, *HeroService* et Location dans le constructeur, en enregistrant leurs valeurs dans des champs privés:

```
constructor(
  private route: ActivatedRoute,
  private heroService: HeroService,
  private location: Location
) {}
```

L'objet *ActivatedRoute* contient des informations sur l'itinéraire vers cette instance de *HeroDetailComponent*. Ce composant s'intéresse au conteneur de paramètres du chemin extrait de l'URL. Le paramètre « *id* » est l'identifiant du héros à afficher.

Le *HeroService* obtient des données de héros du serveur distant et ce composant l'utilisera pour obtenir le héros à afficher.

L'emplacement est un service Angular pour interagir avec le navigateur. Vous l'utiliserez plus tard pour revenir à la vue qui a navigué ici.

Extraire le paramètre du chemin de l'*id*

Dans le hook de cycle de vie *ngOnInit()*, appelez *getHero()* et définissez-le comme suit.

```
ngOnInit(): void {
  this.getHero();
}

getHero(): void {
  const id = +this.route.snapshot.paramMap.get('id');
  this.heroService.getHero(id)
    .subscribe(hero => this.hero = hero);
}
```

Route.snapshot est une image statique des informations du chemin peu après la création du composant.

Le *paramMap* est un dictionnaire de valeurs de paramètres de route extraites de l'URL. La clé « id » renvoie l'identifiant *id* du héros à récupérer.

Les paramètres du chemin sont toujours des chaînes. L'opérateur *JavaScript(+)* convertit la chaîne en un nombre, ce qui devrait être un identifiant *id* du héros.

Le navigateur se rafraîchit et l'application se bloque avec une erreur de compilation. *HeroService* n'a pas de méthode *getHero()*. Ajoutez-le maintenant.

Ajouter *HeroService.getHero()*

Ouvrez *HeroService* et ajoutez cette méthode *getHero()*

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

```
getHero(id: number): Observable<Hero> {
  // TODO: send the message _after_ fetching the hero
  this.messageService.add(`HeroService: fetched hero id=${id}`);
  return of(HEROES.find(hero => hero.id === id));
}
```

Notez bien que ce sont les backticks *(`)* qui définissent un ***modèle de libellé*** JavaScript pour l'intégration de l'*ID*.

Faites vos essais

Le navigateur est actualisé et l'application fonctionne à nouveau. Vous pouvez cliquer sur un héros dans le tableau de bord ou dans la liste des héros et accéder à la vue détaillée de ce héros.

Si vous collez *localhost:4200/detail/11* dans la barre d'adresse du navigateur, le routeur accède à la vue détaillée du héros avec l'*ID:11*, « Mr. Nice ».

Trouver le chemin du retour

En cliquant sur le bouton de retour du navigateur, vous pouvez revenir à la liste des héros ou à la vue du tableau de bord, en fonction de ce qui vous a envoyé dans la vue détaillée.

Ce serait bien d'avoir un bouton sur la vue *HeroDetail* qui peut le faire.

Ajoutez un bouton de retour au bas du modèle de composant et liez-le à la méthode *goBack()* du composant.

src/app/hero-detail/hero-detail.component.html (bouton de retour)

```
<button (click)="goBack()">go back</button>
```

Ajoutez une méthode *goBack()* à la classe de composant qui recule d'une étape dans la pile d'historique du navigateur à l'aide du service Location que vous avez précédemment injecté.

src/app/hero-detail/hero-detail.component.ts (goBack)

```
goBack(): void {
  this.location.back();
}
```

Actualisez le navigateur et commencez à cliquer. Les utilisateurs peuvent naviguer dans l'application, du tableau de bord aux détails des héros et vice-versa, de la liste des héros aux mini détails aux détails des héros et de nouveau aux héros.

Vous avez satisfait à toutes les exigences de navigation qui ont propulsé cet article.

Revue Des Codes Finals

Voici les fichiers de code discutés sur cette page et votre application devrait ressembler à cet ***exemple en direct*** / ***télécharger l'exemple***.

1-AppRoutingModule

src/app/app-routing.module.ts

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

import { DashboardComponent }   from './dashboard/dashboard.component';
import { HeroesComponent }      from './heroes/heroes.component';
import { HeroDetailComponent }  from './hero-detail/hero-detail.component';

const routes: Routes = [
  { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
  { path: 'dashboard', component: DashboardComponent },
  { path: 'detail/:id', component: HeroDetailComponent },
  { path: 'heroes', component: HeroesComponent }
];

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

2-AppModule

src/app/app.module.ts

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

```
import { AppComponent }          from './app.component';
import { DashboardComponent }    from './dashboard/dashboard.component';
import { HeroDetailComponent }   from './hero-detail/hero-detail.component';
import { HeroesComponent }       from './heroes/heroes.component';
import { MessagesComponent }     from './messages/messages.component';

import { AppRoutingModule }      from './app-routing.module';

@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    AppRoutingModule
  ],
  declarations: [
    AppComponent,
    DashboardComponent,
    HeroesComponent,
    HeroDetailComponent,
    MessagesComponent
  ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

3-HeroService

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);
  }

  getHero(id: number): Observable<Hero> {
    // TODO: send the message _after_ fetching the hero
    this.messageService.add(`HeroService: fetched hero id=${id}`);
    return of(HEROES.find(hero => hero.id === id));
  }
}
```

4-AppComponent

src/app/app.component.html

```
<h1>{{title}}</h1>
<nav>
  <a routerLink="/dashboard">Dashboard</a>
  <a routerLink="/heroes">Heroes</a>
</nav>
<router-outlet></router-outlet>
<app-messages></app-messages>
```

src/app/app.component.css

```
/* AppComponent's private CSS styles */
```

```
h1 {
  font-size: 1.2em;
  color: #999;
  margin-bottom: 0;
}
h2 {
  font-size: 2em;
  margin-top: 0;
  padding-top: 0;
}
nav a {
  padding: 5px 10px;
  text-decoration: none;
  margin-top: 10px;
  display: inline-block;
  background-color: #eee;
  border-radius: 4px;
}
nav a:visited, a:link {
  color: #607D8B;
}
nav a:hover {
  color: #039be5;
  background-color: #CFD8DC;
}
nav a.active {
  color: #039be5;
}
```

5-DashboardComponent

src/app/dashboard/dashboard.component.html

```
<h3>Top Heroes</h3>
<div class="grid grid-pad">
  <a *ngFor="let hero of heroes" class="col-1-4"
      routerLink="/detail/{{hero.id}}">
    <div class="module hero">
      <h4>{{hero.name}}</h4>
    </div>
  </a>
</div>
```

src/app/dashboard/dashboard.component.ts

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

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

  constructor(private heroService: HeroService) { }

  ngOnInit() {
    this.getHeroes();
  }

  getHeroes(): void {
    this.heroService.getHeroes()
      .subscribe(heroes => this.heroes = heroes.slice(1, 5));
  }
}
```

src/app/dashboard/dashboard.component.css

```
/* DashboardComponent's private CSS styles */
[class*='col-'] {
  float: left;
  padding-right: 20px;
  padding-bottom: 20px;
}
[class*='col-']:last-of-type {
  padding-right: 0;
}
a {
  text-decoration: none;
}
*, *:after, *:before {
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
}
h3 {
  text-align: center; margin-bottom: 0;
}
h4 {
  position: relative;
}
.grid {
  margin: 0;
}
.col-1-4 {
  width: 25%;
}
.module {
  padding: 20px;
  text-align: center;
  color: #eee;
  max-height: 120px;
  min-width: 120px;
  background-color: #607D8B;
  border-radius: 2px;
}
.module:hover {
  background-color: #EEE;
  cursor: pointer;
  color: #607d8b;
}
.grid-pad {
  padding: 10px 0;
}
.grid-pad > [class*='col-']:last-of-type {
  padding-right: 20px;
}
@media (max-width: 600px) {
  .module {
    font-size: 10px;
    max-height: 75px; }
}
@media (max-width: 1024px) {
  .grid {
    margin: 0;
  }
  .module {
    min-width: 60px;
  }
}
```

6-HeroesComponent

src/app/heroes/heroes.component.html

```
<h2>My Heroes</h2>
<ul class="heroes">
  <li *ngFor="let hero of heroes">
    <a routerLink="/detail/{{hero.id}}">
      <span class="badge">{{hero.id}}</span> {{hero.name}}
```

```
    </a>
  </li>
</ul>
```

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 {
  heroes: Hero[];

  constructor(private heroService: HeroService) { }

  ngOnInit() {
    this.getHeroes();
  }

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

src/app/heroes/heroes.component.css

```
/* HeroesComponent's private CSS styles */
.heroes {
  margin: 0 0 2em 0;
  list-style-type: none;
  padding: 0;
  width: 15em;
}
.heroes li {
  position: relative;
  cursor: pointer;
  background-color: #EEE;
  margin: .5em;
  padding: .3em 0;
  height: 1.6em;
  border-radius: 4px;
}

.heroes li:hover {
  color: #607D8B;
  background-color: #DDD;
  left: .1em;
}

.heroes a {
  color: #888;
  text-decoration: none;
  position: relative;
  display: block;
  width: 250px;
}

.heroes a:hover {
  color:#607D8B;
}

.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;
  min-width: 16px;
  text-align: right;
  margin-right: .8em;
  border-radius: 4px 0 0 4px;
}
```

7-HeroDetailComponent

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>
  <button (click)="goBack()">go back</button>
</div>
```

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

```
import { Component, OnInit, Input } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Location } from '@angular/common';

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

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

  constructor(
    private route: ActivatedRoute,
    private heroService: HeroService,
    private location: Location
  ) {}

  ngOnInit(): void {
    this.getHero();
  }

  getHero(): void {
    const id = +this.route.snapshot.paramMap.get('id');
    this.heroService.getHero(id)
      .subscribe(hero => this.hero = hero);
  }

  goBack(): void {
    this.location.back();
  }
}
```

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

```
/* HeroDetailComponent's private CSS styles */
label {
```

```
  display: inline-block;
  width: 3em;
  margin: .5em 0;
  color: #607D8B;
  font-weight: bold;
}
input {
  height: 2em;
  font-size: 1em;
  padding-left: .4em;
}
button {
  margin-top: 20px;
  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: #ccc;
  cursor: auto;
}
```

Récapitulons!

Dans cet article, nous avons pu:

- ajouter le routeur Angular pour naviguer parmi les différents composants.
- transformer *AppComponent* en un shell de navigation avec des liens *<a>* et un *<routeur-outlet>*.
- configurer le routeur dans le module *AppRoutingModule*.
- définir des chemins simples, un chemin de redirection et un chemin paramétré.
- utiliser la directive *routerLink* dans les éléments d'ancrage.
- refaçonner une vue héros/détail étroitement couplée dans une vue détaillée routée.
- partager le *HeroService* parmi plusieurs composants.