GANATAN

Routing avec angular 20

Guide Complet

Mise à jour du 29 mai 2025

Le **Routing** (ou **Routage**) permet de gérer les **URLs** et les **routes** d'une application web.

Dans ce tutoriel, nous allons **intégrer** le routing en utilisant **Angular 20.0.0**.

Angular dispose de la bibliothèque **@angular/router**, qui permet de mettre en place facilement et rapidement une navigation efficace au sein d'une application Angular.

Comment le faire ?

Pour réaliser ce routing voici un résumé de ce que nous allons faire

- **Avant de Commencer**
  Qu'est-ce que le routing ou routage ?
  Avec un petit exemple de la vie

- **Création de notre projet Angular**
  Nous utiliserons un projet existant contenant les fonctionnalités essentielles.
  Le projet a été généré avec Angular CLI.

- **Structure du projet**
  Avant d'intégrer nos modifications il nous faut choisir une structure.

- **Initialisation du projet**
  En utilisant angular CLI

- **Child Routes**
  Comment gèrer des routes plus complexes

- **Effectuer les Tests**
  Réaliser les tests unitaires.

- **Code source**
  Pour les plus pressés d'entre vous, le code complet du projet est disponible sur Github.

  Il vous suffit de vous rendre à l'adresse suivante
  https://github.com/ganatan/angular-react-routing

Avant de commencer

Tout d'abord un exemple pour essayer de comprendre la théorie.

Vous aimez probablement le cinéma.

C'est le week-end et vous hésitez entre deux films.

Du moins 2 films vous intéressent ou plutôt intéressent vos enfants (à vous de voir qui possèdent la télécommande).

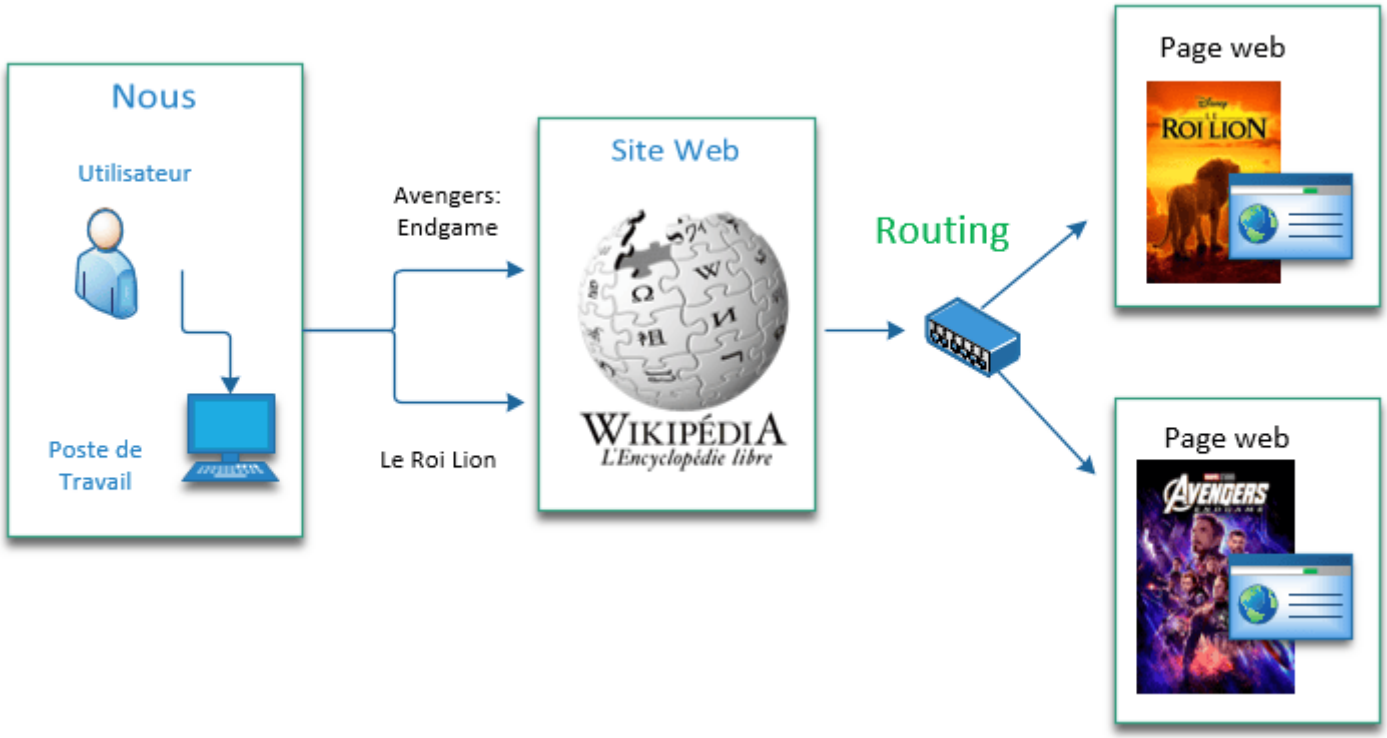
Un ordinateur , un smartphone ou une tablette et le wifi, et c'est parti.

Wikipedia devient notre ami et nous donne un résumé.

C'est rapide et c'est simple.

Au niveau technique voyons ce que cela donne en image.

Le routing avec Wikipedia



Au final tout dépendra comment vous voyez la vie.

Mais Wikipedia devient un chef de gare qui vous dira ou aller.

Et il vaut mieux un chef de gare plutôt costaud vu que vous ne serez pas le seul à lui poser des questions.

La vie c'est une comédie ou c'est une tragédie ?

Quelques explications maintenant.

Un site web est constitué d'une multitude de pages.
Ces pages sont écrites grâce notamment au langage **HTML**.
HTML signifie **HyperText Markup Language**.

L'**hypertexte** est la technologie qui permettra de relier une page à d'autres pages via des **hyperliens**.

Le **Routing** est le mécanisme qui permet de naviguer d'une page à une autre dans notre exemple.

Par exemple si vous tapez ces deux url dans votre navigateur.

- https://fr.wikipedia.org/wiki/Le_Roi_lion
- https://fr.wikipedia.org/wiki/Avengers:_Endgame

En fonction du nom de film indiqué dans l'url, l'application Web de Wikipedia va

déterminer le traitement à effectuer.

Ce traitement permettra d'afficher la page web correspondante au film demandé (ici **Le_Roi_lion** ou **Avengers:_Endgame**).

C'est ce que l'on appelle le **Routage** (traduit littéralement en anglais par **Routing**).

Si nous devions faire une comparaison c'est un peu comme un **aiguilleur du ciel** ou un **chef de gare**.

Sans Routing on ne sait plus on va.

Donc vous l'aurez compris sans routing pas d'application Web.

Ce qui pourrait donner ça.

Ce principe de fonctionnement étant omniprésent dans l'utilisation d'un site web, il nous faut l'appliquer au plus vite dans tout projet web.

Nous allons donc implémenter ce mécanisme dans notre projet Angular.

Avertissement

Donc évidemment fini de rigoler.
La partie technique commence.

Si vous n'y connaissez rien en angular il vaut mieux s'accrocher.

C'est pour ça que je vous détaille tout le process.
Et au passage je vous donne le code complet à implémenter au fur et à mesure.

Sinon IA or not IA où est le vrai **Wild Bill Kelso**

> **je m'appelle Buffalo bill kelso quand c'est la guerre je fais la guerre**

Création du projet Angular

Pour pouvoir continuer ce tutoriel nous devons bien évidemment disposer de certains éléments

- **Node.js** : La plateforme javascript
- **Git** : Le logiciel de gestion de versions.
- **Angular CLI** : L'outil fourni par Angular.
- **Visual Studio code** : Un éditeur de code.

Vous pouvez consulter le tutoriel suivant qui vous explique en détails comment faire

- https://www.ganatan.com/tutorials/demarrer-avec-angular

Dans ce tutoriel nous allons utiliser un projet existant dont les caractéristiques sont

- Genéré avec Angular CLI

```
# Créez un répertoire demo (le nom est ici arbitraire)
mkdir demo

# Allez dans ce répertoire
cd demo

# Récupérez le code source sur votre poste de travail
git clone https://github.com/ganatan/angular-react-starter

# Allez dans le répertoire qui a été créé
cd angular-react-starter
cd frontend-angular

# Exécutez l'installation des dépendances (ou librairies)
npm install

# Exécutez le programme
npm run start

# Vérifiez son fonctionnement en lançant dans votre navigateur la commande
http://localhost:4200/
```

Structure du projet

Lors de la création du projet avec Angular CLI la structure créée par défaut est la suivante

Remarque
Le répertoire environments n'est plus généré automatiquement depuis Angular 15.

Je l'ai rajouté manuellement dans le tutoriel précédent **Démarrer avec Angular** .

Ce qui donne une arborescence de base, **100% Angular** comme celle-ci.

Arborescence de base

```
|-- src/
    |-- app
    |-- assets
    |-- environments
package.json
```

Je vous propose d'adapter l'arborescence proposée.
Le choix que je fais est arbitraire ça veut dire que je fais ce que je veux.
Mais ne prenez pas ça pour argent comptant votre structure pourra prendre n'importe quelle autre forme.
Ce choix sera néanmoins suivi dans les tutoriels suivants.
Donc je vous l'explique.

- **core** regroupera les éléments que l'on retrouvera dans tous les projets.
- **features** regroupera les éléments spécifiques à une application.
- **shared** regroupera les composants réutilisables.

Arborescence adaptée

```
|-- app
    |-- core
    |-- features
        |-- about
        |-- contact
        |-- home
        |-- login
        |-- not-found
        |-- signup
    |-- shared
|-- assets
|-- environments
```

A propos d'architecture je vous conseille la video de cet excellent youtuber **Gaëtan Rouziès**

Qui nous livre ici ses réflexions

https://www.youtube.com/watch?v=JUFTGXUjnuE

Un avis différent

Initialisation

Un rapide tour d'horizon avant de voir les détails techniques
Angular propose une librairie **angular@routeur** qui permet de gérer le routing.

Dans les versions Antérieures à Angular 17 le routing était gérée de la façon suivante.

La librairie angular@routeur était importée dans un **module dédié** au routing.
Ce module portait le nom suivant **app-routing.module.ts.**

Depuis la version 17 le routing est implicite lors de la création du projet avec Angular CLI.

Le routing est ainsi géré via 3 fichiers de base.

- **app.ts**
- **app.config.ts**
- **app.routes.ts**

Le schéma suivant vous expliquera comment il est géré avec la version 17.

Tout au long de ce didacticiel nous essaierons d'utiliser les commandes que propose angular-cli pour genérer automatiquement le code source nécessaire.

Tout d'abord nous allons créer six composants qui seront utilisés dans le routing.
Un composant correspondra à une page Web.

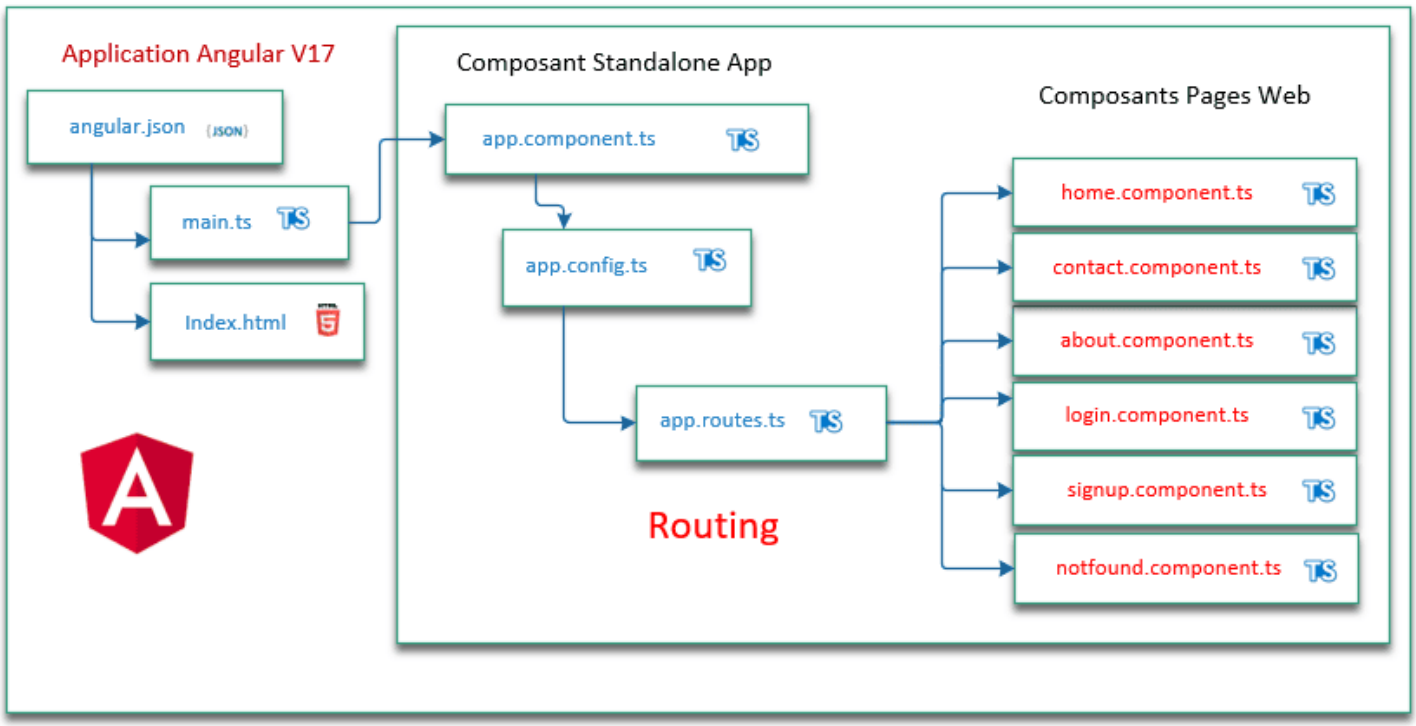
Ce sont six pages Web classiques que l'on retrouve dans tout site web.

- **Home**
- **Contact**
- **About**
- **Login**
- **Signup**
- **NotFound**

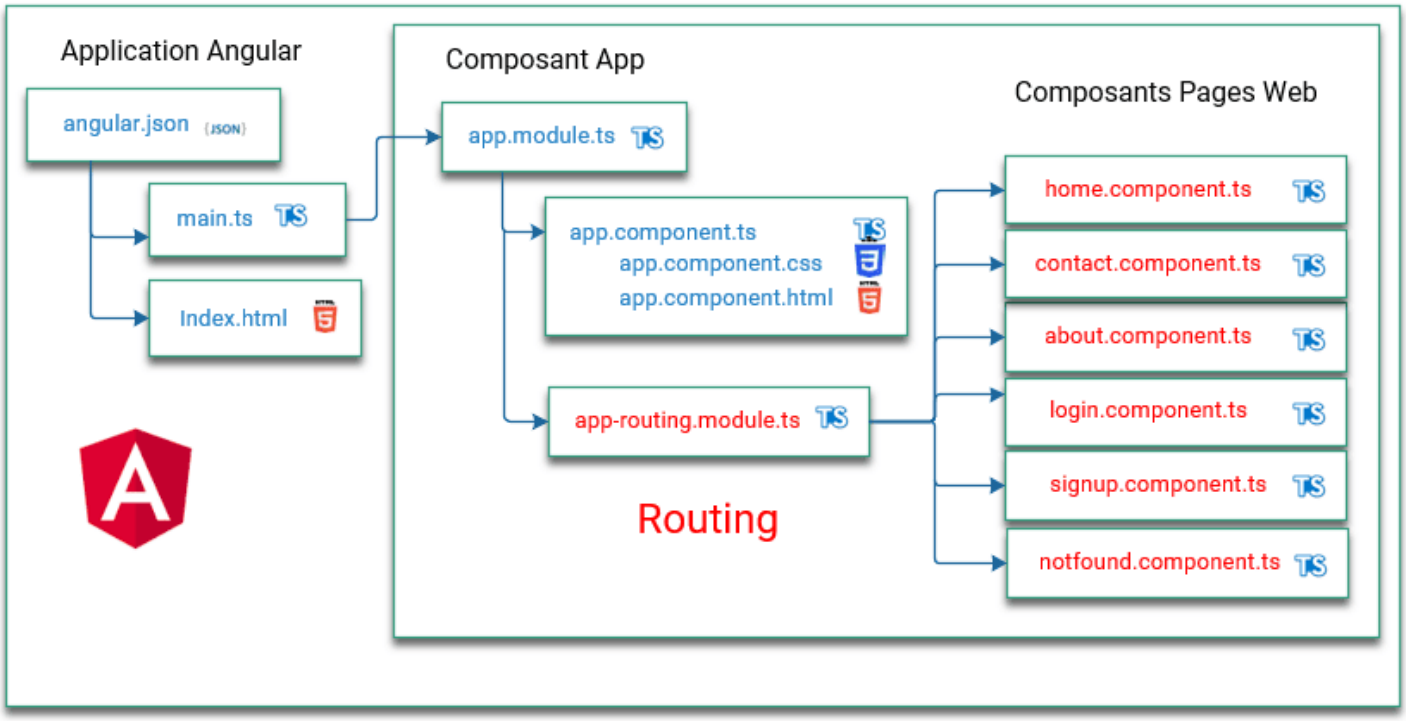
Donc deux schémas pour récapituler ce que nous allons faire avec la version 17

et sa comparaison avec les versions 16 et antérieures.

Routing avec Angular 17



Routing avec Angular 16



Nous utiliserons angular CLI pour cela.

Le tutoriel suivant vous donnera des informations sur les commandes Angular CLI

https://www.ganatan.com/tutorials/demarrer-avec-angular-cli

Exécutons les commandes suivantes.

ng g correspond à la commande **ng generate**

```
# Création des nouveaux composants (méthode 1)
ng generate component features/contact
ng generate component features/login
ng generate component features/signup
ng generate component features/home
ng generate component features/about
ng generate component features/not-found

# Création des nouveaux composants (méthode 2)
ng g c features/contact
ng g c features/login
ng g c features/signup
ng g c features/home
ng g c features/about
ng g c features/not-found
```

Le répertoire **features** est créé par la commande éxécutée.

Tous les fichiers relatifs à chaque composant sont créés automatiquement par angular CLI.

Par exemple pour le composant **Home** 4 fichiers sont créés

- **home.css** (code CSS dédié au design)
- **home.html** (code HTML)
- **home.spec.ts** (code de test unitaire du composant)
- **home.ts** (partie logique en typescript de notre composant)

Il faut ensuite modifier les fichiers suivants

- **app-routes.ts**
- **styles.css**
- **app.html**
- **app.css**
- **app.ts**
- **app.spec.ts**

Ce qui permettra de traiter le routing désiré et les composants appelés.

src/app/app-routes.ts

```
import  { Routes } from '@angular/router';

import  { Home } from './features/home/home';

import  { Login } from './features/login/login';
import  { Signup } from './features/signup/signup';
import  { NotFound } from './features/not-found/not-found';

import  { About } from './features/about/about';
import  { Contact } from './features/contact/contact';

export  const routes: Routes = [
    { path: '', component: Home, },

    { path: 'login', component: Login },
    { path: 'signup', component: Signup },

    { path: 'about', component: About },
    { path: 'contact', component: Contact },

    { path: '**', component: NotFound }
];
```

src/styles.css

```
body {
  color : black;
  font-weight :  400 }
```

Dans le composant AppComponent nous prendrons soin de rajouter l'élément **router-outlet** au niveau du fichier app.component.html.

Il indiquera au router ou afficher les éléments graphiques routés.

L'élément **routerLink** permettra de créer le lien vers les pages souhaitées.

src/app/app.html

```
<div class="app">
  <header>
    <section>
      <h1>{{ title }}</h1>
    </section>
    <nav>
      <h2>3 Links with Routes</h2>
      <ul>
        <li><a routerLink="/">Home</a></li>
        <li><a routerLink="/login">Login</a></li>
        <li><a routerLink="/signup">Signup</a></li>
      </ul>
      <h3>2 Links with Child Routes</h3>
      <ul>
        <li><a routerLink="/about">About</a></li>
        <li><a routerLink="/contact">Contact</a></li>
      </ul>
    </nav>
  </header>

  <main>
    <h4>Routes Result</h4>
    <router-outlet></router-outlet>
  </main>

  <footer>
    <a href="{{ footerUrl }}">{{ footerLink }}</a>
  </footer>

</div>
```

src/app.css

```
h1 {
   color : blue;
}

.app {
font-family: Arial, Helvetica, sans-serif;
     max- width : 500px;
   margin : auto;
}
```

src/app.ts

```
import  { Component } from '@angular/core';
import  { CommonModule } from '@angular/common';
import  { RouterLink, RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  imports: [CommonModule, RouterLink, RouterOutlet],
  templateUrl: './app.html',
  styleUrl: './app.css'
})
export class App {
  title = 'angular-routing';
  footerUrl = 'https://www.ganatan.com';
  footerLink = 'www.ganatan.com';
}
```

Le routing fonctionne

Après toutes ces modifications vérifions que cela fonctionne.
On éxécute le script de lancement et on teste le résultat dans le navigateur.

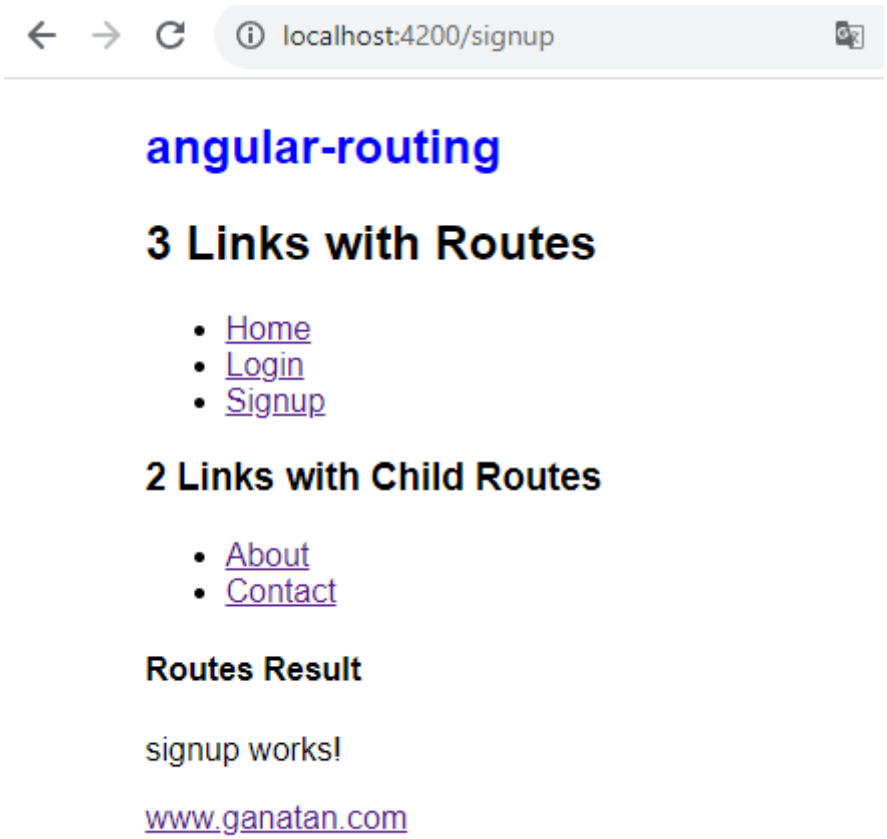
```
# Exécutez l'application
npm run start

# Testez l'application dans votre navigateur
http://localhost:4200
```

Le programme fonctionne , testez le routing en cliquant sur les différents liens.

On est d'accord ce n'est pas très joli.
Mais on a fait rapide, on a utilisé du HTML et du CSS très simples.

On peaufinera tout ça plus tard en utilisant **Bootstrap** dans un tutoriel suivant.

En attendant le résultat en image.



Pour que les tests fonctionnent

Il nous faut rajouter le module **RouterTestingModule** au niveau des tests unitaires du composant App.

Ce rajout se fait au niveau du fichier de test correspondant **app.component.spec.ts**.

src/app/app.spec.ts

```
import  { TestBed } from '@angular/core/testing';
import  { App } from './app';
import  { ActivatedRoute } from '@angular/router';

describe ('App', () => {
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [App],
      providers: [
        {
          provide: ActivatedRoute,
          useValue: {}
        }
      ]
    }).compileComponents();
  });

  it('should create the app', () => {
    const fixture = TestBed.createComponent(App);
    const app = fixture.componentInstance;
    expect(app).toBeTruthy();
  });

  it('should render title', () => {
    const fixture = TestBed.createComponent(App);
    fixture.detectChanges();
    const compiled = fixture.nativeElement as HTMLElement;
    expect(compiled.querySelector('h1')?.textContent).toContain('angular-routing');
  });
});
```

Child Routes

Pour gérer des routes plus complexes, par exemple faire du routing à partir du composant contact, il nous faut aborder la gestion des Child routes ou Nesting Routes.

Cette question est évoquée dans la documentation
https://angular.io/guide/router#child-route-configuration

Nous allons rajouter cinq composants qui seront accessibles à partir des composants Contact et About

- **mailing (pour contact)**
- **mapping (pour contact)**
- **website (pour contact)**

- **skill (pour about)**
- **experience (pour about)**

```
# Rajout des trois composants pour contact
ng generate component features/contact/mailing
ng generate component features/contact/mapping
ng generate component features/contact/website


# Rajout des deux composants pour about
ng generate component features/about/skill
ng generate component features/about/experience
```

4 fichiers sont automatiquement créés pour chaque composant par exemple pour mailing

- **mailing.css**
- **mailing.html**
- **mailing.spec.ts**
- **mailing.ts**

Il nous reste à modifier le routing

- **contact.html**
- **contact.ts**
- **contact.spec.ts**
- **about.html**
- **about.ts**
- **about.spec.ts**

- **app-routes.ts**

contact.html

```
<div>
   <p>contact works!</p>
   <ul>
     <li><a routerLink="/contact/mailing">Mailing</a></li>
     <li><a routerLink="/contact/mapping">Mapping</a></li>
     <li><a routerLink="/contact/website">Website</a></li>
   </ul>
   <h4>Child Routes Result</h4>
   <router-outlet></router-outlet>
 </div>
```

contact.ts

```
import  { Component } from '@angular/core';
import  { CommonModule } from '@angular/common';
import  { RouterLink, RouterOutlet } from '@angular/router';


@Component({
  selector: 'app-contact',
  imports: [CommonModule, RouterLink, RouterOutlet],
  templateUrl: './contact.html',
  styleUrl: './contact.css'
})
export class Contact {

}
```

about.component.html

```
<div>
   <p>about works!</p>
   <ul>
     <li><a routerLink="/about/experience">experience</a></li>
     <li><a routerLink="/about/skill">skill</a></li>
   </ul>
   <h4>Child Routes Result</h4>
   <router-outlet></router-outlet>
  </div>
```

about.ts

```
import  { Component } from '@angular/core';
import  { CommonModule } from '@angular/common';
import  { RouterLink, RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-about',
  imports: [CommonModule, RouterLink, RouterOutlet],
  templateUrl: './about.html',
  styleUrl: './about.css'
})
 export class About {

}
```

app-routes.ts

```
import { Routes } from '@angular/router';

import { Home } from './features/home/home';
import { Login } from './features/login/login';
import { Signup } from './features/signup/signup';
import { NotFound } from './features/not-found/not-found';
import { About } from './features/about/about';
import { Contact } from './features/contact/contact';

import { Experience } from './features/about/experience/experience';
import { Skill } from './features/about/skill/skill';

import { Mailing } from './features/contact/mailing/mailing';
import { Mapping } from './features/contact/mapping/mapping';
import { Website } from './features/contact/website/website';

export const routes: Routes = [
  { path: '', component: Home, },

  { path: 'login', component: Login },
  { path: 'signup', component: Signup },

  {
    path: 'about', component: About,
    children: [
      { path: '', component: Experience },
      { path: 'experience', component: Experience },
      { path: 'skill', component: Skill },
    ],
  },
  {
    path: 'contact', component: Contact,
    children: [
      { path: '', component: Mailing },
      { path: 'mailing', component: Mailing },
      { path: 'mapping', component: Mapping },
      { path: 'website', component: Website },
    ],
  },

  { path: '**', component: NotFound }
];
```

contact.spec.ts

```
import  { ComponentFixture, TestBed } from '@angular/core/testing';

import  { Contact } from './contact';
import  { ActivatedRoute } from '@angular/router';

describe ('Contact', () => {
  let component: Contact;
  let fixture: ComponentFixture<Contact>;

   beforeEach(async () => {
     await TestBed.configureTestingModule({
       imports: [Contact],
       providers: [
         {
           provide: ActivatedRoute,
           useValue: {}
         }
       ]
     })
     .compileComponents();

     fixture = TestBed.createComponent(Contact);
     component = fixture.componentInstance;
     fixture.detectChanges();
   });

   it('should create', () => {
     expect(component).toBeTruthy();
   });
 });
```

about.component.spec.ts

```
import  { ComponentFixture, TestBed } from '@angular/core/testing';

import  { About } from './about';
import  { ActivatedRoute } from '@angular/router';

describe ('About', () => {
  let component: About;
  let fixture: ComponentFixture<About>;

   beforeEach(async () => {
     await TestBed.configureTestingModule({
       imports: [About],
       providers: [
         {
           provide: ActivatedRoute,
           useValue: {}
         }
       ]
     }).compileComponents();

     fixture = TestBed.createComponent(About);
     component = fixture.componentInstance;
     fixture.detectChanges();
   });

   it('should create', () => {
     expect(component).toBeTruthy();
   });
});
```

Il ne reste plus qu'à tester les scripts suivants.

```
# Développement
npm run start
http://localhost:4200/

# Test de la page not-found
http://localhost:4200/lien-inconnu

# Tests
npm run test

# Production
npm run build
```

Code source

Le code source utilisé en début de tutoriel est disponible sur github
https://github.com/ganatan/angular-react-starter

Le code source obtenu à la fin de ce tutoriel est disponible sur github
https://github.com/ganatan/angular-react-routing

Et si ça vous a plu, toute l'équipe attend normalement une étoile sur github.

WTF pas d'étoile sur github !

L' étape suivante est la gestion du Lazy Loading.

Elle nécessite un tutoriel complet qui est à l'adresse suivante

- Etape 3 : Lazy loading avec Angular

Les étapes suivantes vous permettront d'obtenir une application prototype.

- Etape 4 : Bootstrap avec Angular
- Etape 5 : Modules avec Angular
- Etape 6 : Server Side Rendering avec angular
- Etape 7 : Progressive Web App avec Angular
- Etape 8 : Search Engine Optimization avec Angular
- Etape 9 : HttpClient avec Angular