Angular : tests unitaires

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Angular

Deux catégories de tests

- tests manuels :
  - coûteux
  - répétitif (corriger et retester)
- tests automatiques (via des programmes)

Angular

Plusieurs niveaux de tests

- tests unitaires
- tests d'intégration
- tests fonctionnels
- tests de charge
- tests d'acceptation

Angular

tests unitaires

- Permettent de tester une **unité (partie) isolée** de l'application (souvent une fonction ou une méthode).
- Indiquent au développeur que le code fait **bien les choses**.

Angular

tests unitaires

- Permettent de tester une **unité (partie) isolée** de l'application (souvent une fonction ou une méthode).
- Indiquent au développeur que le code fait **bien les choses**.

Remarque

Les tests qui communiquent avec une base de données ou une **API** externe ne sont pas considérés comme tests unitaires, mais plutôt comme tests d'intégration.

Angular

tests d'intégration

- permettent de vérifier que les **unités isolées** fonctionnent correctement ensemble.
- peuvent nécessiter de composants extérieurs (Service web, base de données...).

Angular

Autres niveaux de tests

- **tests fonctionnels** (de bout-en-bout ou **end-to-end** en anglais) : sont écrits du point de vue de l'utilisateur final depuis l'interface utilisateur, pour vérifier que l'application répond aux exigences.
  Ils indiquent au développeur que le code fait **les bonnes choses**.

- **tests d'acceptation** : permettent de vérifier que l'**application** respecte bien le besoin fonctionnel.

- **tests de charge** (système) : permettent de vérifier si un système peut gérer une charge spécifiée : le nombre d'utilisateurs simultanés...

Angular

Et les tests de non-régression ?

Ils permettent de vérifier que la livraison de nouvelles fonctionnalités n'aura pas d'effet de bord sur les fonctionnalités existantes (programme préalablement testé).

Angular

Autres tests

- **tests de performance**
- **tests de fiabilité**
- ...

Angular

Tests unitaires avec **Angular** : outils

- **Karma** : moteur de test
- **Jasmine** : langage d'assertion permettant de
  - décrire les objets à tester
  - exécuter des commandes avant et après chaque test
  - évaluer le bon fonctionnement de notre code
- **angular-cli** : interface fournissant des commandes (comme `ng test`) permettant de lancer les tests.

Angular

Les fichiers de test

- ayant l'extension `.spec.ts`
- généré automatiquement à la création d'un projet, composant, service...

Angular

Principales fonctions de **Jasmine**

- `Describe` : description de l'objet à tester
- `beforeEach` et `afterEach` : exécution de code avant ou après chaque test
- `it` : bloc de description de la fonctionnalité à tester
- `expect` : évaluation du cas à tester

Angular

Principales fonctions de `Expect` comporte les matchers de comparaison suivants

- `toBe` : pour vérifier l'égalité stricte (`===` du JavaScript)
- `toEqual` : pour vérifier l'égalité non-stricte
- `toContain` : pour vérifier qu'un `Array` ou une `string` contient un élément
- `toBeDefined` : pour vérifier si un objet est défini
- `toBeNull` : pour vérifier si la valeur est nulle
- `toBeTruthy` ou `toBeFalsy` : pour vérifier si la valeur est `true` ou `false`
- `toHaveBeenCalled` : pour vérifier si une méthode a été appelée
- `toHaveBeenCalledWith` : pour vérifier si une méthode a été appelée avec des paramètres d'une certaine valeur

Angular

Contenu de la librairie standard de tests **@angular/core/testing**

- La fonction `async` : permet à **Jasmine** de ne pas passer au test suivant que lorsque tous les traitements asynchrones seront terminés.
- L'objet `TestBed` : permet de configurer le module de test.

Angular

Contenu de la librairie standard de tests **@angular/core/testing**

- La fonction `async` : permet à **Jasmine** de ne pas passer au test suivant que lorsque tous les traitements asynchrones seront terminés.
- L'objet `TestBed` : permet de configurer le module de test.

Rôle de la librairie standard de tests **@angular/core/testing**

permet d'initier un module de test et d'indiquer les directives, services, composants, etc à tester.

Angular

Pour créer un nouveau projet Angular

```
ng new test-angular
```

Angular

Pour créer un nouveau projet Angular

```
ng new test-angular
```

Ou le raccourci

```
ng n test-angular
```

Angular

Modifions `app.component.ts`

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

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

Et `app.component.html`

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
</div>
```

Angular

Modifions le contenu de notre classe de test `app.component.spec.ts`

```
import { AppComponent } from './app.component';

describe('AppComponent', () => {
  it('should create the app', () => {
    const app = new AppComponent()
    expect(app).toBeTruthy();
  });

  it(`should have as title 'test-angular'`, () => {
    const app = new AppComponent()
    expect(app.title).toEqual('test-angular');
  });
});
```

Angular

Modifions le contenu de notre classe de test `app.component.spec.ts`

```
import { AppComponent } from './app.component';

describe('AppComponent', () => {
  it('should create the app', () => {
    const app = new AppComponent()
    expect(app).toBeTruthy();
  });

  it(`should have as title 'test-angular'`, () => {
    const app = new AppComponent()
    expect(app.title).toEqual('test-angular');
  });
});
```

Explication

- `describe` : permet de regrouper des spécifications (tests) (Utilisée souvent par les frameworks de test comme **Jasmine** et **Jest**)
- `it` : permet de déclarer une spécification (test)
- `expect` : assertion

Angular

Pour lancer les tests

```
ng test
```

Angular

Pour lancer les tests

```
ng test
```

Résultat

```
2 specs, 0 failures, randomized with seed 04057
  AppComponent
    should have as title 'test-angular'
    should create the app
```

Angular

Dans le code précédent

- Dans chaque test, on instanciait le composant avec l'opérateur `new` (comme tout objet en POO).
- Pour crée une instance du composant dans un environnement de test **Angular** avec :
  - la gestion du cycle de vie
  - l'injection de dépendances
  - la détection des changements
- Il est recommandé d'utiliser `TestBed`

Modifions le contenu de notre classe de test `app.component.spec.ts`

```
import { TestBed } from '@angular/core/testing';
import { AppComponent } from './app.component';

describe('AppComponent', () => {

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

  it(`should have as title 'test-angular'`, () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    expect(app.title).toEqual('test-angular');
  });
});
```

Modifions le contenu de notre classe de test `app.component.spec.ts`

```
import { TestBed } from '@angular/core/testing';
import { AppComponent } from './app.component';

describe('AppComponent', () => {

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

  it(`should have as title 'test-angular'`, () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    expect(app.title).toEqual('test-angular');
  });
});
```

Explication

- `TestBed.createComponent` : permet de créer le composant
- `fixture.componentInstance` : permet d'obtenir une instance de ce composant

Angular

Vérifier le résultat des tests

```
2 specs, 0 failures, randomized with seed 04057
  AppComponent
    should have as title 'test-angular'
    should create the app
```

Angular

Dans certains cas

- Avant de démarrer un test, il faut faire certains traitements :
  - instancier un objet de la classe,
  - se connecter/ouvrir une ressource,
- Après le test, il faut aussi fermer certaines ressources

Angular

Dans certains cas

- Avant de démarrer un test, il faut faire certains traitements :
  - instancier un objet de la classe,
  - se connecter/ouvrir une ressource,
- Après le test, il faut aussi fermer certaines ressources

Solution

Utiliser les méthodes `beforeAll()`, `beforeEach()`, `afterAll()` et `afterEach()`

Angular

Ajoutons les quatre méthodes suivantes dans `describe()`

```
beforeAll(() => {
  console.log('beforeAll');
})
beforeEach(() => {
  console.log('beforeEach');
})
afterAll(() => {
  console.log('afterAll');
})
afterEach(() => {
  console.log('afterEach');
})
```

Angular

Relancez le test et vérifiez le résultat suivant

```
beforeAll
beforeEach
afterEach
beforeEach
afterEach
afterAll
```

Angular

Relancez le test et vérifiez le résultat suivant

```
beforeAll
beforeEach
afterEach
beforeEach
afterEach
afterAll
```

Explication

- `beforeAll` : exécutée seulement avant le premier test
- `afterAll` : exécutée seulement après le dernier test
- `beforeEach` : exécutée avant chaque test
- `afterEach` : exécutée après chaque test

Angular

Utilisons la méthode `beforeEach` **pour restructurer** `app.component.spec.ts`

```
import { TestBed } from '@angular/core/testing';
import { AppComponent } from './app.component';

describe('AppComponent', () => {

  let fixture = null;
  let app:AppComponent;

  beforeEach(() => {
    fixture = TestBed.createComponent(AppComponent);
    app = fixture.componentInstance;
  })

  it('should create the app', () => {
    expect(app).toBeTruthy();
  });

  it(`should have as title 'test-angular'`, () => {
    expect(app.title).toEqual('test-angular');
  });
});
```

Angular

Créons un service `calcul`

```
ng g s services/calcul
```

Angular

Créons un service `calcul`

```
ng g s services/calcul
```

Résultat

```
CREATE src/app/services/calcul.service.spec.ts (357 bytes)
CREATE src/app/services/calcul.service.ts (135 bytes)
```

Angular

Ajoutons les méthodes suivantes dans `calcul.service.ts`

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

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

  constructor() { }

  somme(a: number, b: number) {
    return a + b;
  }

  division(a: number, b: number) {
    return a / b;
  }
}
```

Angular

Code généré pour `calcul.service.spec.ts`

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

import { CalculService } from './calcul.service';

describe('CalculService', () => {
  let service: CalculService;

  beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(CalculService);
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });
});
```

Angular

Code généré pour `calcul.service.spec.ts`

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

import { CalculService } from './calcul.service';

describe('CalculService', () => {
  let service: CalculService;

  beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(CalculService);
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });
});
```

Explication

TestBed.inject(CalculService) : un service on ne l'instancie pas, mais on l'injecte.

Angular

Testons les méthodes `somme` **et** `division` **dans** `calcul.service.spec.ts`

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

import { CalculService } from './calcul.service';

describe('CalculService', () => {
  let service: CalculService;

  beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(CalculService);
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });
  it('should add two numbers', () => {
    expect(service.somme(2, 3)).toBe(5);
  });
  it('should divide two numbers', () => {
    expect(service.division(10, 5)).toBe(2);
  });
  it('should divide by zero', () => {
    expect(service.division(10, 0)).toBePositiveInfinity();
  });
});
```

Angular

Vérifier le résultat des tests

```
6 specs, 0 failures, randomized with seed 72592
  CalculService
    should add two numbers
    should divide two numbers
    should be created
    should divide by zero
  AppComponent
    should have as title 'test-angular'
    should create the app
```

Angular

Créons un composant `calcul`

```
ng g c components/calcul
```

Angular

Créons un composant `calcul`

```
ng g c components/calcul
```

Résultat

```
CREATE src/app/components/calcul/calcul.component.html (21 bytes)
CREATE src/app/components/calcul/calcul.component.spec.ts (559 bytes)
CREATE src/app/components/calcul/calcul.component.ts (202 bytes)
CREATE src/app/components/calcul/calcul.component.css (0 bytes)
UPDATE src/app/app.module.ts (486 bytes)
```

Angular

Commençons par injecter `CalculService` **dans** `calcul.component.ts`

```
import { Component } from '@angular/core';
import { CalculService } from 'src/app/services/calcul.service';

@Component({
  selector: 'app-calcul',
  templateUrl: './calcul.component.html',
  styleUrls: ['./calcul.component.css']
})
export class CalculComponent {

  constructor(private calculService: CalculService) { }

}
```

Angular

Définissons ensuite la méthode `sommeCarre`

```
import { Component } from '@angular/core';
import { CalculService } from 'src/app/services/calcul.service';

@Component({
  selector: 'app-calcul',
  templateUrl: './calcul.component.html',
  styleUrls: ['./calcul.component.css']
})
export class CalculComponent {

  constructor(private calculService: CalculService) { }

  sommeCarre(x: number, y: number) {
    return this.calculService.somme(x * x, y * y)
  }
}
```

Angular

Contenu de `calcul.component.spec.ts`

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { CalculComponent } from './calcul.component';

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

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [CalculComponent]
    });
    fixture = TestBed.createComponent(CalculComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

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

Angular

Question

Comment vérifier que la méthode de service sera appelée par le composant ?

Angular

Question

Comment vérifier que la méthode de service sera appelée par le composant ?

Solution

Utiliser les Spy.

Angular

Commençons par déclarer le service dans `calcul.component.spec.ts`

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { CalculComponent } from './calcul.component';

describe('CalculComponent', () => {
  let component: CalculComponent;
  let fixture: ComponentFixture<CalculComponent>;
  let calculService: CalculService;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [CalculComponent]
    });
    fixture = TestBed.createComponent(CalculComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

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

Angular

Ajoutons la spécification suivante dans `calcul.component.spec.ts`

```
it('should call somme of CalculService', () => {
    spyOn(calculService, 'somme');
    component.sommeCarre(2, 3);
    expect(calculService.somme).toHaveBeenCalled();
});
```

Angular

Ajoutons la spécification suivante dans `calcul.component.spec.ts`

```
it('should call somme of CalculService', () => {
    spyOn(calculService, 'somme');
    component.sommeCarre(2, 3);
    expect(calculService.somme).toHaveBeenCalled();
});
```

Explication

- `spyOn(calculService, 'somme')` : on demande d'espionner la méthode `somme` de `CalculService`
- `component.sommeCarre(2, 3)` : on exécute la méthode `sommeCarre`
- `expect(calculService.somme).toHaveBeenCalled()` : on vérifie que la méthode `somme` a été appelée

Angular

Vérifier le résultat des tests

```
8 specs, 0 failures, randomized with seed 16342
  CalculComponent
    should create
    should call somme of CalculService
  AppComponent
    should have as title 'test-angular'
    should create the app
  CalculService
    should be created
    should divide two numbers
    should divide by zero
    should add two numbers
```

Angular

Ajoutons la spécification suivante dans `calcul.component.spec.ts`

```
it('should return the squared sum', () => {
    const spyCalcul : jasmine.Spy = spyOn(calculService, 'somme')
    spyCalcul.withArgs(9, 16).and.returnValue(25);
    const result = component.sommeCarre(3, 4);
    expect(result).toEqual(25);
});
```

Angular

Ajoutons la spécification suivante dans `calcul.component.spec.ts`

```
it('should return the squared sum', () => {
    const spyCalcul : jasmine.Spy = spyOn(calculService, 'somme')
    spyCalcul.withArgs(9, 16).and.returnValue(25);
    const result = component.sommeCarre(3, 4);
    expect(result).toEqual(25);
});
```

Explication

- `const spyCalcul : jasmine.Spy = spyOn(calculService, 'somme')` : on crée un espion sur la méthode `somme`
- `spyCalcul.withArgs(9, 16).and.returnValue(25)` : on spécifie à l'espion qu'il doit retourner `25` si la méthode `somme` est sollicitée avec les paramètres 9 et 16.

Angular

Vérifier le résultat des tests

```
9 specs, 0 failures, randomized with seed 22235
  CalculService
    should be created
    should add two numbers
    should divide two numbers
    should divide by zero
  AppComponent
    should have as title 'test-angular'
    should create the app
  CalculComponent
    should create
    should call somme of CalculService
    should return the squared sum
```

Angular

Créons un deuxième fichier `calcul.component.integration.spec.ts` **pour les tests d'intégration et copions sa structure depuis** `calcul.component.spec.ts`

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { CalculComponent } from './calcul/calcul.component';
import { CalculService } from '../services/calcul.service';

describe('CalculComponent (Integration)', () => {
    let component: CalculComponent;
    let fixture: ComponentFixture<CalculComponent>;
    let calculService: CalculService;

    beforeEach(() => {
        TestBed.configureTestingModule({
            declarations: [CalculComponent],
            providers: [CalculService]
        }).compileComponents();

        fixture = TestBed.createComponent(CalculComponent);
        component = fixture.componentInstance;
        calculService = TestBed.inject(CalculService);
    });
});
```

Angular

Sans Spy, vérifions que le composante t le service fonctionnent ensemble

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { CalculComponent } from './calcul/calcul.component';
import { CalculService } from '../services/calcul.service';

describe('CalculComponent (Integration)', () => {
    let component: CalculComponent;
    let fixture: ComponentFixture<CalculComponent>;
    let calculService: CalculService;

    beforeEach(() => {
        TestBed.configureTestingModule({
            declarations: [CalculComponent],
            providers: [CalculService]
        }).compileComponents();

        fixture = TestBed.createComponent(CalculComponent);
        component = fixture.componentInstance;
        calculService = TestBed.inject(CalculService);
    });

    it('should return the squared sum', () => {
        const result = component.sommeCarre(3, 4);
        expect(result).toEqual(25);
    });
});
```

Angular

Vérifier le résultat des tests

```
10 specs, 0 failures, randomized with seed 90877
  CalculComponent (Integration)
    should return the squared sum
  AppComponent
    should create the app
    should have as title 'test-angular'
  CalculService
    should divide two numbers
    should divide by zero
    should be created
    should add two numbers
  CalculComponent
    should call somme of CalculService
    should return the squared sum
    should create
```

Angular

Modifions le composant `calcul.component.ts`

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { CalculService } from 'src/app/services/calcul.service';

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

  x: number = 0
  y: number = 0
  result: number = 0

  constructor(private calculService: CalculService, private route: ActivatedRoute) { }

  ngOnInit(): void {
    this.route.params.subscribe(res => {
      this.x = Number(res['x'])
      this.y = Number(res['y'])
      this.result = this.sommeCarre(this.x, this.y)
    })
  }

  sommeCarre(x: number, y: number) {
    return this.calculService.somme(x * x, y * y)
  }
}
```

Angular

Affichons tous les attributs dans `calcul.component.html`

```
<h1>Résultat</h1>
<p>{{ x }}<sup>2</sup> + {{ y }}<sup>2</sup> = {{ result }}</p>
```

Angular

Affichons tous les attributs dans `calcul.component.html`

```
<h1>Résultat</h1>
<p>{{ x }}<sup>2</sup> + {{ y }}<sup>2</sup> = {{ result }}</p>
```

Sans oublier de définir une route avec les deux paramètres `x` **et** `y` **dans** `app-routing.module.ts`

```
const routes: Routes = [
  { path: 'calcul/:x/:y', component: CalculComponent }
];
```

Préparons `calcul.component.spec.ts` afin de tester la récupération de paramètres

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { CalculComponent } from './calcul.component';
import { CalculService } from 'src/app/services/calcul.service';
import { ActivatedRoute } from '@angular/router';
import { from } from 'rxjs';

describe('CalculComponent', () => {
  let component: CalculComponent;
  let fixture: ComponentFixture<CalculComponent>;
  let calculService: CalculService;
  let route: ActivatedRoute;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [CalculComponent],
      providers: [
        CalculService,
        {
          provide: ActivatedRoute,
          useValue: {
            params: from([{ x: 3, y: 4 }]),
          },
        },
      ]
    });
    calculService = TestBed.inject(CalculService);
    route = TestBed.inject(ActivatedRoute);
    fixture = TestBed.createComponent(CalculComponent);
    component = fixture.componentInstance;
  });
  // les tests précédents
});
```

Angular

Ajoutons la spécification suivante dans `calcul.component.spec.ts`

```
it('should set x and y from route parameters on initialization', () =>
   {
     expect(component.x).toEqual(0);
     expect(component.y).toEqual(0);
     spyOn(route.params, 'subscribe').and.callThrough();
     fixture.detectChanges();
     expect(route.params.subscribe).toHaveBeenCalled();
     expect(component.x).toEqual(3);
     expect(component.y).toEqual(4);
});
```

Angular

Ajoutons la spécification suivante dans `calcul.component.spec.ts`

```
it('should set x and y from route parameters on initialization', () =>
   {
     expect(component.x).toEqual(0);
     expect(component.y).toEqual(0);
     spyOn(route.params, 'subscribe').and.callThrough();
     fixture.detectChanges();
     expect(route.params.subscribe).toHaveBeenCalled();
     expect(component.x).toEqual(3);
     expect(component.y).toEqual(4);
});
```

Explication

- `spyOn(route.params, 'subscribe').and.callThrough()` : on crée un espion et on demande l'exécution du `subscribe` réel lorsque ce dernier est appelé
- `fixture.detectChanges()` : permet d'exécuter les méthodes hooks (`ngOnInit()`...)

Angular

Vérifier le résultat des tests

```
11 specs, 0 failures, randomized with seed 49156
  CalculComponent
    should call somme of CalculService
    should return the squared sum
    should set x and y from route parameters on initialization
    should create
  CalculService
    should divide by zero
    should be created
    should add two numbers
    should divide two numbers
  AppComponent
    should create the app
    should have as title 'test-angular'
  CalculComponent (Integration)
    should return the squared sum
```

Angular

Ajoutons une deuxième spécification pour vérifier que le résultat final est correct

```
it('should update result property after getting parameters', () => {
    const expectedResult = 25;
    const spyCalcul: jasmine.Spy = spyOn(calculService, 'somme')
    spyCalcul.withArgs(9, 16).and.returnValue(25);
    spyOn(route.params, 'subscribe').and.callThrough();
    fixture.detectChanges();
    expect(component.result).toBe(expectedResult)
});
```

Angular

Vérifier le résultat des tests

```
11 specs, 0 failures, randomized with seed 49156
  CalculComponent
    should call somme of CalculService
    should return the squared sum
    should set x and y from route parameters on initialization
    should create
    should update result property after getting parameters
  CalculService
    should divide by zero
    should be created
    should add two numbers
    should divide two numbers
  AppComponent
    should create the app
    should have as title 'test-angular'
  CalculComponent (Integration)
    should return the squared sum
```

Sans Spy, vérifions que le composante et le service fonctionnent ensemble

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { CalculComponent } from './calcul/calcul.component';
import { CalculService } from '../services/calcul.service';
import { ActivatedRoute } from '@angular/router';
import { from } from 'rxjs';

describe('CalculComponent (Integration)', () => {
    let component: CalculComponent;
    let fixture: ComponentFixture<CalculComponent>;
    let calculService: CalculService;
    let route: ActivatedRoute;

    beforeEach(() => {
        TestBed.configureTestingModule({
            declarations: [CalculComponent],
            providers: [
                CalculService,
                {
                    provide: ActivatedRoute,
                    useValue: {
                        params: from([{ x: 3, y: 4 }]),
                    },
                },
            ]
        }).compileComponents();
        fixture = TestBed.createComponent(CalculComponent);
        component = fixture.componentInstance;
        calculService = TestBed.inject(CalculService);
        route = TestBed.inject(ActivatedRoute);
    });
    // + les tests
});
```

Angular

Sans Spy, vérifions que le composante et le service fonctionnent ensemble

```
it('should return the squared sum', () => {
     const result = component.sommeCarre(3, 4);
     expect(result).toEqual(25);
});

it('should return the squared sum after getting params', () => {
     spyOn(route.params, 'subscribe').and.callThrough();
     fixture.detectChanges()
     expect(component.result).toEqual(25);
});
```

Angular

Vérifier le résultat des tests

```
13 specs, 0 failures, randomized with seed 81194
  CalculComponent (Integration)
    should return the squared sum after getting params
    should return the squared sum
  CalculComponent
    should create
    should set x and y from route parameters on initialization
    should update result property after getting parameters
    should return the squared sum
    should call somme of CalculService
  CalculService
    should divide by zero
    should be created
    should divide two numbers
    should add two numbers
  AppComponent
    should create the app
    should have as title 'test-angular'
```

Angular

Créons un fichier `db.json`, à la racine du projet `test-angular`, et qui va nous servir de base de données

```
{
    "personnes": [
        {
            "nom": "wick",
            "prenom": "john",
            "id": 1
        },
        {
            "nom": "green",
            "prenom": "bob",
            "id": 2
        }
    ]
}
```

Angular

Lancer le serveur

```
json-server -p 5555 db.json
```

Angular

Lancer le serveur

```
json-server -p 5555 db.json
```

Résultat (parmi toutes les lignes affichées)

```
Resources http://localhost:5555/personnes
```

Angular

Commençons par créer l'interface `personne`

```
ng g i interfaces/personne
```

Angular

Commençons par créer l'interface `personne`

```
ng g i interfaces/personne
```

Considérons le contenu suivant pour l'interface `Personne`

```
export interface Personne {
    id?: number;
    nom?: string;
    prenom?: string;
}
```

Angular

Créons un service `personne`

```
ng g s services/personne
```

Angular

Créons un service `personne`

```
ng g s services/personne
```

Résultat

```
CREATE src/app/services/personne.service.spec.ts (367 bytes)
CREATE src/app/services/personne.service.ts (137 bytes)
```

Angular

Importons `HttpClientModule` **pour utiliser** `HttpClient`

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';

//+ les autres imports

@NgModule({
  declarations: [
        // les composants
  ],
  imports: [
    BrowserModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Angular

Faisons une première requête HTTP pour récupérer la liste des personnes

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Personne } from '../interfaces/personne';
import { Observable } from 'rxjs';

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

  url = 'http://localhost:5555/personnes/';

  constructor(private http: HttpClient) {

  }
  getAll(): Observable<Personne[]> {
    return this.http.get<Personne[]>(this.url);
  }

}
```

Angular

Créons un composant `personne`

```
ng g c components/personne
```

Angular

Créons un composant `personne`

```
ng g c components/personne
```

Résultat

```
CREATE src/app/components/personne/personne.component.html (23 bytes)
CREATE src/app/components/personne/personne.component.spec.ts (573
bytes)
CREATE src/app/components/personne/personne.component.ts (210 bytes)
CREATE src/app/components/personne/personne.component.css (0 bytes)
UPDATE src/app/app.module.ts (587 bytes)
```

Pour récupérer la liste des personnes dans `personne.component.ts`, il faut s'abonner

```
import { Component, OnInit } from '@angular/core';
import { Personne } from '../../interfaces/personne';
import { PersonneService } from '../../services/personne.service';

@Component({
  selector: 'app-personne',
  templateUrl: './personne.component.html',
  styleUrls: ['./personne.component.css']
})
export class PersonneComponent implements OnInit {
  personne: Personne = {};
  personnes: Array <Personne> = [];

  constructor(private personneService: PersonneService) {  }

  ngOnInit(): void {
    this.personneService.getAll().subscribe(res => {
      this.personnes = res;
    });
  }
}
```

Angular

Le template `personne.component.html` où on affiche le résultat

```
<h2>Liste des personnes</h2>
<ul>
  <li *ngFor="let elt of personnes">
    {{ elt.prenom }}  {{ elt.nom }}
  </li>
</ul>
```

Angular

Et enfin la route

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { CalculComponent } from './components/calcul/calcul.component';
import { PersonneComponent } from './components/personne/personne.
   component';

const routes: Routes = [
  { path: 'calcul/:x/:y', component: CalculComponent },
  { path: 'personne', component: PersonneComponent }
];

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

Angular

Contenu de `personne.service.spec.ts`

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

import { PersonneService } from './personne.service';

describe('PersonneService', () => {
  let service: PersonneService;

  beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(PersonneService);
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });
});
```

Contenu de `personne.component.spec.ts`

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

import { PersonneComponent } from './personne.component';

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

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [PersonneComponent],

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

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

Angular

Avant d'écrire les spécifications, commençons par configurer `personne.service.spec.ts`

```
import { TestBed } from '@angular/core/testing';
import { HttpClientTestingModule, HttpTestingController } from '@angular/common/http/testing';
import { PersonneService } from './personne.service';

describe('PersonneService', () => {
  let service: PersonneService;
  let http: HttpTestingController;

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpClientTestingModule],
      providers: [PersonneService]
    });

    service = TestBed.inject(PersonneService);
    http = TestBed.inject(HttpTestingController);
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });

});
```

Angular

Et aussi `personne.component.spec.ts`

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { PersonneComponent } from './personne.component';
import { PersonneService } from '../../services/personne.service';
import { HttpClientTestingModule } from '@angular/common/http/testing';

describe('PersonneComponent', () => {
  let component: PersonneComponent;
  let fixture: ComponentFixture<PersonneComponent>;
  let personneService: PersonneService;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [PersonneComponent],
      imports: [HttpClientTestingModule],
      providers: [PersonneService]
    }).compileComponents();

    fixture = TestBed.createComponent(PersonneComponent);
    component = fixture.componentInstance;
    personneService = TestBed.inject(PersonneService);
  });

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

});
```

Angular

Explication

- `HttpClientTestingModule` ≡ `HttpClientModule` pour le mode test.
- `HttpTestingController` : classe qui permet de simuler l'envoie de requête **HTTP** vers le serveur.

Ajoutons la spécification suivante dans `personne.service.spec.ts`

```
it('should retrieve all personnes from the API via GET', () => {
    const mockPersonnes: Personne[] = [
      { id: 1, nom: 'John', prenom: 'Doe' },
      { id: 2, nom: 'Jane', prenom: 'Smith' }
    ];

    service.getAll().subscribe(personnes => {
      expect(personnes).toEqual(mockPersonnes);
    });

    const req = http.expectOne('http://localhost:5555/personnes/');
    expect(req.request.method).toBe('GET');

    req.flush(mockPersonnes);
});
```

Ajoutons la spécification suivante dans `personne.service.spec.ts`

```
it('should retrieve all personnes from the API via GET', () => {
    const mockPersonnes: Personne[] = [
      { id: 1, nom: 'John', prenom: 'Doe' },
      { id: 2, nom: 'Jane', prenom: 'Smith' }
    ];

    service.getAll().subscribe(personnes => {
      expect(personnes).toEqual(mockPersonnes);
    });

    const req = http.expectOne('http://localhost:5555/personnes/');
    expect(req.request.method).toBe('GET');

    req.flush(mockPersonnes);
});
```

Explication

- `http.expectOne('http://localhost:5555/personnes/')` :intercepte l'envoi de la requête **HTTP** et retourne les détails de la requête (stockés dans `req`)
- `req.flush(mockPersonnes)` : permet de simuler la réponse à cette requête et de vérifier qu'elle contient les données de `mockPersonnes`

Angular

Et la spécification suivante dans `personne.component.spec.ts`

```
it('should fetch personnes on init', () => {
    const personnes: Personne[] = [
        { id: 1, nom: 'John', prenom: 'Doe' },
        { id: 2, nom: 'Jane', prenom:'Smith' }
    ];
    spyOn(personneService, 'getAll').and.returnValue(of(personnes));

    fixture.detectChanges()

    expect(personneService.getAll).toHaveBeenCalled();
    expect(component.personnes).toEqual(personnes);
});
```

Angular

Vérifier le résultat des tests

```
17 specs, 0 failures, randomized with seed 89314
  CalculService
    should divide two numbers
    should be created
    should add two numbers
    should divide by zero
  PersonneService
    should be created
    should retrieve all personnes from the API via GET
  AppComponent
    should create the app
    should have as title 'test-angular'
  CalculComponent
    should return the squared sum
    should set x and y from route parameters on initialization
    should create
    should call somme of CalculService
    should update result property after getting parameters
  CalculComponent (Integration)
    should return the squared sum
    should return the squared sum after getting params
  PersonneComponent
    should create
    should fetch personnes on init
```

Angular

Créons un service `calcul`

```
ng e2e
No > Cypress Nightwatch WebdriverIO
The package @cypress/schematic@2.5.0 will be installed and
executed.
Would you like to proceed? Yes

Would you like the default `ng e2e` command to use Cypress? Yes

Would you like to add Cypress component testing? This will add
all files needed for Cypress component testing. No
```

Angular

Résultat

```
CREATE cypress.config.ts (134 bytes)
CREATE cypress/tsconfig.json (139 bytes)
CREATE cypress/e2e/spec.cy.ts (143 bytes)
CREATE cypress/fixtures/example.json (85 bytes)
CREATE cypress/support/commands.ts (1377 bytes)
CREATE cypress/support/e2e.ts (649 bytes)
UPDATE package.json (1195 bytes)
UPDATE angular.json (3772 bytes)
 Packages installed successfully.
```