Angular

Créer des applications web front-end

LES TESTS SUR UNE APP ANGULAR

Les objectifs du testing :

- Clarifier ce que fait le code en testant son comportement dans différentes situations
- Éviter que du code nouveau ne produise des erreurs
- Optimiser l'architecture du code (le code doit au maximum être testable)

```
"test": {
  "builder": "@angular-devkit/build-angular:karma",
  "options": {
    "main": "src/test.ts",
    "polyfills": "src/polyfills.ts",
    "tsConfig": "tsconfig.spec.json",
    "karmaConfig": "karma.conf.js",
    "inlineStyleLanguage": "scss",
    "assets": [
      "src/favicon.ico",
      "src/assets"
    ],
    "styles": [
      "src/styles.scss"
    ],
    "scripts": []
  }
}
```

1 Les frameworks de tests

Angular propose principalement 3 frameworks de test

- **Jasmine** permet d'écrire les tests unitaires
- **Karma** permet d'exécuter ces tests dans le navigateur
- **Protractor** permet d'écrire des tests end-to-end pour simuler le comportement utilisateur dans le navigateur

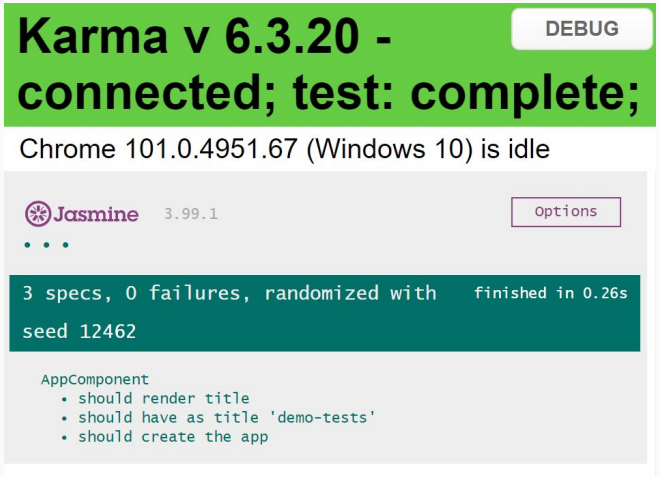
L'environnement de test permet de tester les classes des composants, ainsi que leurs interactions avec la vue HTML

2 Notre premier test

- Créer une application Angular
- Ouvrir le dossier de l'application dans VS Code

Depuis le terminal, exécuter la commande `ng test`

Ceci exécutera les tests qui se trouvent dans app.component.spec.ts



3 Lancer les tests dans le navigateur

ng test

Cette commande exécute tous les fichiers de test Jasmine *.spec.ts (cette configuration provient de test.ts)

Karma exécutera les tests dans l'environnement du navigateur

ng test `--watch=false`

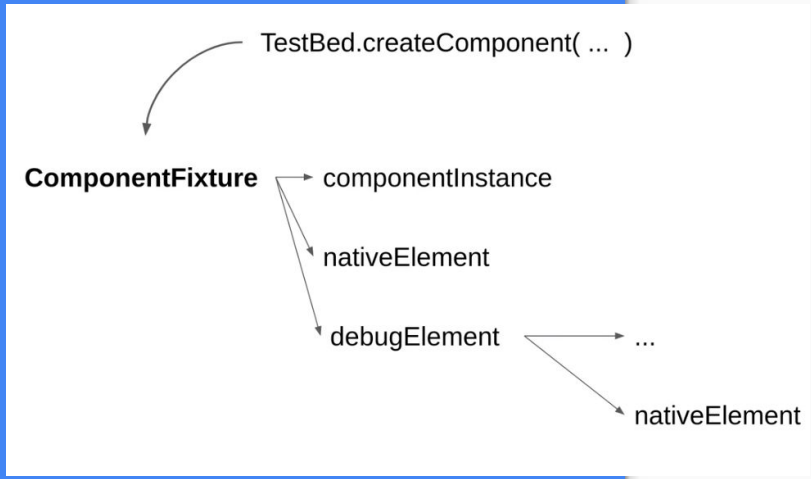
Cette option supprime le livereload des tests à chaque sauvegarde de fichier

ng test `--browser=ChromeHeadless`

Cette option supprime le livereload des tests à chaque sauvegarde de fichier

https://angular.io/cli/test

4 Quelques testing utilities



L'objet TestBed nous permet d'aller plus loin… en créant une fixture

- TestBed.createComponent()
  permet de créer un objet particulier, nommé une fixture. Cet objet expose non seulement l'instance du component que l'on teste, mais aussi d'autres propriétés particulières.

- fixture.componentInstance
  c'est l'instance du component que l'on veut tester

- fixture.debugElement
  renvoie un objet qui contient beaucoup de propriétés. et méthodes utiles pour les tests

- fixture.nativeElement
  renvoie l'objet du template HTML du component (nous permettant d'accéder au DOM du component)

5 Les éléments d'un fichier de test

```
1  import { TestBed } from '@angular/core/testing';
2  import { RouterTestingModule } from '@angular/router/testing';
3  import { AppComponent } from './app.component';
4
5  describe('AppComponent', () => {
6    beforeEach(async () => {
7      await TestBed.configureTestingModule({
8        imports: [
9          RouterTestingModule
10       ],
11       declarations: [
12         AppComponent
13       ],
14     }).compileComponents();
15   });
16
17   it('should create the app', () => {
18     const fixture = TestBed.createComponent(AppComponent);
19     const app = fixture.componentInstance;
20     expect(app).toBeTruthy();
21   });
22
23 });
```

Dans le fichier app.component.spec.ts, par exemple:
Nous retrouvons 3 parties principales

- TestBed
  Cet objet expose des méthodes qui permettent d'instancier un module de test, et compiler ses components. Un component instancié par un module de test est identique à un component instancié par le module de l'application (ngModule)

- describe(NomDuComponent, callback)
  contient notre notre bloc de tests en callback

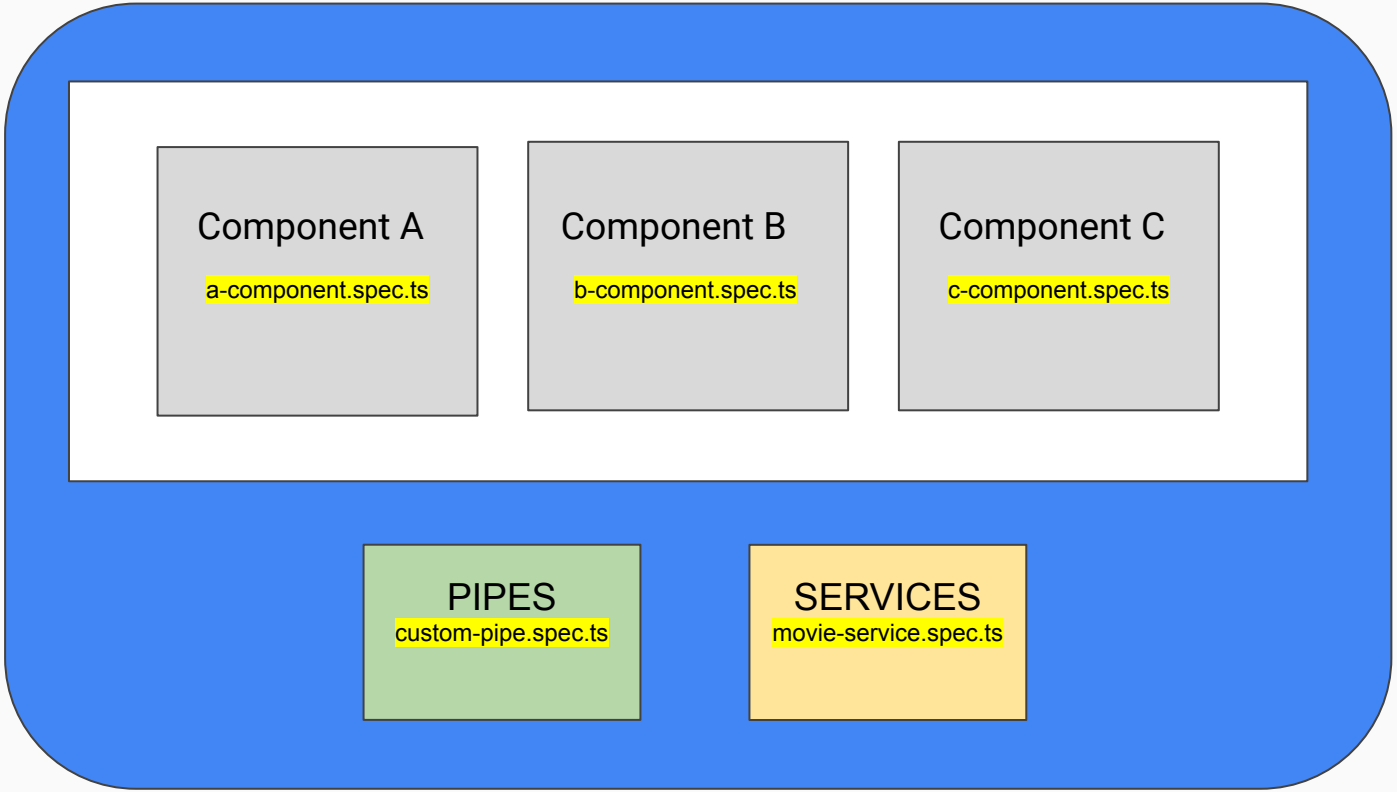
- it('description du test', callback)
  permet d'écrire un test

- expect(param).matcherFunction()
  C'est la méthode qui permet de tester ce qui est attendu

6 LES DIFFÉRENTS TESTS QUE NOUS POUVONS METTRE EN PLACE

Les tests concernent chaque partie de l'application :

- Les Components
- Les Pipes
- Les Services



7 Exemple : test d'un component sans dépendance

```
/*
  la méthode describe,
  contient une suite de tests unitaires
*/
describe('AppComponent', () => {
  let fixture:ComponentFixture<AppComponent>;
  let comp:AppComponent;
  let elt : HTMLElement;
  let debug: DebugElement;
  /*
  BeforeEach()
  reéxécutera l'environnement
  avant chaque test (it())
  */
  beforeEach(() => {
    /*
      On DECLARE L'ENVIRONNEMENT DES TESTS
      de la suite de tests unitaires,
      puis compiler les components
    */
    TestBed.configureTestingModule({
      declarations: [
        AppComponent,
        CapitalizePipe,
        StarsPipe
      ]
    }
    ).compileComponents()

    fixture = TestBed.createComponent(AppComponent);
    comp = fixture.componentInstance;
    elt = fixture.nativeElement;
    debug = fixture.debugElement;
  })
```

Une fois que l'on a défini l'environnement, …
nous écrivons un test unitaire :

```
// it() permet de décrire un test unitaire
 it('title should be Demo', () => {
   // On active la détection de changement dans l'environnement de test
   fixture.detectChanges();
   /*
    attendu 1 : la propriété title du component = 'demo'
   */
   expect(comp.title).toEqual('demo');
   /*
   attendu 2 :
   le texte de la view doit inclure la string
   de la propriété 'title'de l'instance du component
   */
   let titleElt = elt.querySelector('h1');
   let pipe = new CapitalizePipe();
   expect(titleElt?.textContent).toContain(pipe.transform(comp.title));

 })
```

8 Exemple : test d'un component avec dépendances

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

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ SearchbarComponent ],
      providers: [
        {
        provide: MovieService,
        useClass:MockMovieService
        }
      ],
      imports: [
        RouterTestingModule,
        //AppRoutingModule
      ],
    })
    .compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(SearchbarComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

```

S'il dépend d'un service, il est préférable de tester un component avec un mock de données

```
/**
  * TEST 1 : si 0 caractères dans le champ de recherche de la vue HTML
  * Attendu : componentInstance.foundMovies = []
*/
 it('should foundMovies=[] if userInput string=0 chars', () => {
   // on simule l'envoi d'un event sur le champ de recherche
   component.searchMoviesAction('');
   // attendu : que foundMovies soit un array vide
   expect(component.foundMovies).toEqual([]);
 })
  /*****************************************************************/
 /** La class MockMovieService sera utilisée à la place du service **/
 class MockMovieService {
   // ...La classe doit contenir un mock de données
   get foundMovies$():Observable<MovieModel[]> {
     return this._foundMovies$;
   }
  }
```

9 Exemple : Test isolé

Ci-dessous nous avons un pipe

```ts
import { Pipe, PipeTransform } from '@angular/core';
/**
 * Le pipe nommé 'stars' est déclaré ici
 * et est utilisable directement dans les vues HTML
 * de l'appliction avec la syntax {{ 4 | stars }}
 */
@Pipe({
  name: 'stars'
})
export class StarsPipe implements PipeTransform {
  /*
    La méthode transform()
    - prend en paramètre la valeur que la vue cherche à afficher
    - renvoie une chaîne de caractère formatée

    Il est alors assez simple de tester un pipe,
    via un test isolé avec Jasmine
  */
  transform(value: number): string {
    let starsTpl:string = '';
    for(let i=1; i<=value;i++) {
      starsTpl+='<i class="fa-solid fa-star"></i>';
    }
    return starsTpl;
  }

}

```

Nous pouvons créer des tests isolés, c'est à dire sans environnement de test comme pour un component, c'est le cas pour tester un pipe

```ts
import { StarsPipe } from './stars.pipe';

describe('StarsPipe', () => {
  it('create an instance', () => {
    const pipe = new StarsPipe();
    expect(pipe).toBeTruthy();
  });

  it('should return stars html template i tags', () => {
    const pipe = new StarsPipe()
    expect(pipe.transform(2)).toBe('<i class="fa-solid
fa-star"></i><i class="fa-solid fa-star"></i>')
  })
});
```