



Test unitaire

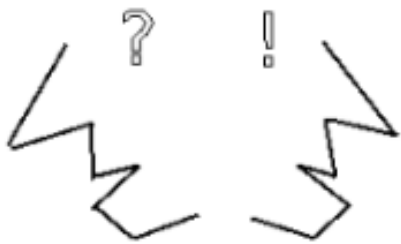
O. Capuozzo

Version 1.0, 2020-11-25

Table des matieres

Présentation	1
Prise en main de la gestion des tests avec Angular	1
Prérequis	1
Contexe Angular	1
Tester un composant	2
CLI ng et frameworks de test	3
Exemple Hello world !	3
<component>.spec.ts initial	5
Lancement des tests	6
Ajout de tests spécifiques	8
Test des valeurs par défaut	8
Test de réactivité avec la vue	9
Exercice	10
Evolution du composant	10
Paramètre de l'application	10
Début de creation du service météo (première version)	12
Update de l'interface Temperature	13
Vérification	17
Conclusion	17
Exercices	17

Présentation



Support de cours sur l'initiation aux tests unitaires

Prise en main de la gestion des tests avec Angular

Prérequis

- Des bases en programmation objet
- Avoir pris en main le framework Angular à travers un tutoriel et avoir réalisé des exercices en autonomie
- Avoir une première expérience dans les tests unitaires

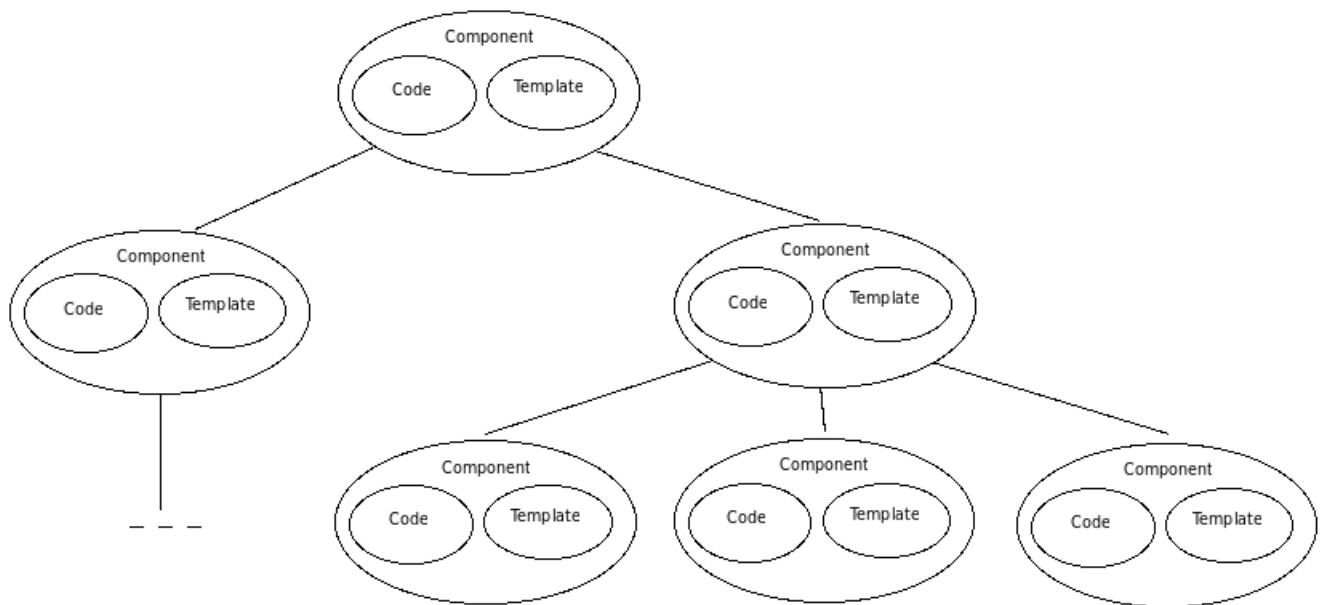
Contexe Angular

La programmation d'application frontend avec Angular (mais pas seulement) est basée sur la notion de *Component* (composant). Les composants pouvant être regroupés en *ngModules* (un package à la Angular).

Un composant regroupe :

- une structure HTML pour son rendu dans la page où il est déclaré
- une logique de comportement (actions, événements) définit par une classe Typescript
- un sélecteur CSS qui définit comme utiliser le composant dans un template parent
- optionnellement, des styles CSS à appliquer au rendu de ce composant

Pour synthétiser, une application frontend basée sur JS/HTML/CSS et le concept de Composant forme un arbre conceptuel de cette forme.



Tester un composant

Le test d'un composant consiste à vérifier si ce dernier se comporte comme attendu.



Le composant testé est appelé *component-under-test* (CUT).

- Le rendu est-il correct ?
- Le comportement du composant, le code, est-il fidèle à nos attentes, à ses spécifications ?

On comprend alors que l'on ne peut tester un tel composant sans un contexte d'environnement.

Au minimum, un contexte de composant est composé de :

- un template "parent" pour le rendu du composant
- une instance de la classe TypeScript liée au composant.

Ces deux problèmes ont leur solution dans l'écosystème Angular.

- Pour interpréter le rendu HTML du composant, on peut soit simuler le comportement d'un navigateur, soit faire directement appel à un navigateur.
- Il est courant que l'instance de la classe TypeScript du composant utilise elle-même des services qui lui sont injectés, ce qui rend plus difficile l'instanciation de cette classe.

Un **CUT** n'a pas besoin d'être injecté avec de vrais services. En fait, il est généralement préférable d'injecter des sortes de "doublures" (*stubs*, *fakes*, *spies*, ou *mocks*). Le but de la spécification est de tester le composant, pas le service, et les services réels peuvent poser problème.

Injecter le vrai service pourrait être un cauchemar. Le service réel peut demander à l'utilisateur des informations de connexion et tenter d'accéder à un serveur d'authentification. Ces comportements peuvent être difficiles à intercepter. Il est beaucoup plus facile et plus sûr de créer et d'enregistrer une doublure à la place du vrai service.

— <https://angular.io/guide/testing-components-scenarios>

TestBed est une classe de Angular, dont le rôle est de configurer et d'initialiser un environnement pour les tests unitaires.

CLI ng et frameworks de test

Par défaut (2020/2021), Angular CLI génère du code de test basé sur *Jasmine* et *Karma*.

- Jasmine, un framework open source pour les tests unitaire en JS : <https://jasmine.github.io/>
- Karma : *A simple tool that allows you to execute JavaScript code in multiple real browsers*. C'est un outil qui génère un serveur Web qui exécute le code source par rapport au code de test pour chacun des navigateurs connectés. Les résultats de chaque test par rapport à chaque navigateur sont examinés et affichés via la ligne de commande au développeur afin qu'il puisse voir quels navigateurs et tests ont réussi ou échoué. <https://github.com/karma-runner/karma/>

Le code des tests, appelé également **spécifications**, est placé dans un fichier situé **dans** le dossier du composant et portant le nom du composant postfixé par **.spec.ts**.

On se réfèrera également aux fichiers de configuration à la racine du projet : **angular.json** et **karma.json**

Exemple Hello world !

Afin d'illustrer les instructions de test, nous créons, dans un projet servant d'exemple et via CLI ng, un composant que nous nommerons **hello-word**.

Listing 1. génération d'un composant à l'aide de cli ng

```
$ ng generate component HelloWorld
CREATE src/app/hello-world/hello-world.component.scss (0 bytes)
CREATE src/app/hello-world/hello-world.component.html (26 bytes)
CREATE src/app/hello-world/hello-world.component.spec.ts (655 bytes)
CREATE src/app/hello-world/hello-world.component.ts (295 bytes)
UPDATE src/app/app.module.ts (690 bytes)
$
```

Modifions le template :

```
<h2 class="title">Hello world !</h2>
<p>Nice day {{name}}.</p>
<p>Today it is {{temperature.temp}}°</p>
```

Et le code lié :

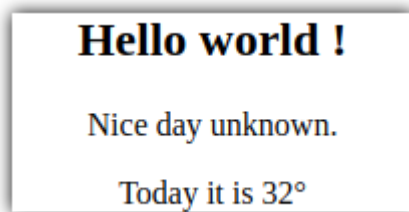
Listing 2. src/app/hello-world/hello-world.component.ts

```
1 import { Component, OnInit } from '@angular/core';
2
3 // une interface est un modèle de structure d'objet.
4 // Tout objet (ou classe) se réfèrent à cette interface devra implémenter
5 // ce modèle.
6 interface Temperature {
7   readonly temp : number
8 }
9
10 @Component({
11   selector: 'app-hello-world',
12   templateUrl: './hello-world.component.html',
13   styleUrls: ['./hello-world.component.scss']
14 })
15 export class HelloWorldComponent implements OnInit {
16
17   name : string = "unknown";
18   temperature : Temperature = {temp : 32};
19
20   constructor() { }
21
22   ngOnInit(): void {
23   }
24 }
```

Le composant principal, parent de notre composant, `app.component.html` (avec des classes CSS de bulma) :

```
<style>
  .centre {
    text-align: center;
  }
</style>
<section class="section centre">
  <div class="container">
    <div class="column ">
      <app-hello-world></app-hello-world>
    </div>
  </div>
</section>
```

à l'exécution nous obtenons :



<component>.spec.ts initial

Via la commande `ng generate component HelloWorld` un script de test est généré : `hello-world.component.spec.ts`.

Dans un premier nous analysons le code généré, puis l'étendrons par ajout de nouveaux tests.

```

1 import { ComponentFixture, TestBed } from '@angular/core/testing'; ①
2
3 import { HelloWorldComponent } from './hello-world.component';
4
5 describe('HelloWorldComponent', () => { ②
6   let component: HelloWorldComponent;
7   let fixture: ComponentFixture<HelloWorldComponent>;
8
9   beforeEach(async () => { ③
10     await TestBed.configureTestingModule({ ④
11       declarations: [ HelloWorldComponent ]
12     })
13     .compileComponents();
14   });
15
16   beforeEach(() => { ⑤
17     fixture = TestBed.createComponent(HelloWorldComponent);
18     component = fixture.componentInstance;
19     fixture.detectChanges();
20   });
21
22   it('should create', function() { ⑥
23     expect(component).toBeTruthy(); ⑦
24   });
25 });

```

① : import de bibliothèques Angular dédiées aux tests

② : Donne un nom à la portée des différents tests *it('xxx')* définis dans ce block, connu sous le nom de **suite** (ou *test suite*)

③ : **beforeEach** : Pour définir les actions à exécuter **avant** chacun des tests dans cette suite.

④ : Déclaration et compilation du composant (CUT - *Component Under Test*). L'appel de la méthode statique **configureTestingModule** de **TestBed** est asynchrone (**promise**), nous demandons d'attendre son retour (**await**) avant créer le composant à l'étape suivante.

⑤ : Initialisation du contexte (instanciation du composant et son contexte)

⑥ : Vérifie que la variable locale *component* a bien été initialisée

⑦ : **toBeTruthy** vérifie que *component* est "not undefined". Une instruction de test de **Jasmine**.

Lancement des tests

La commande CLI pour le lancement des tests est : **ng test**

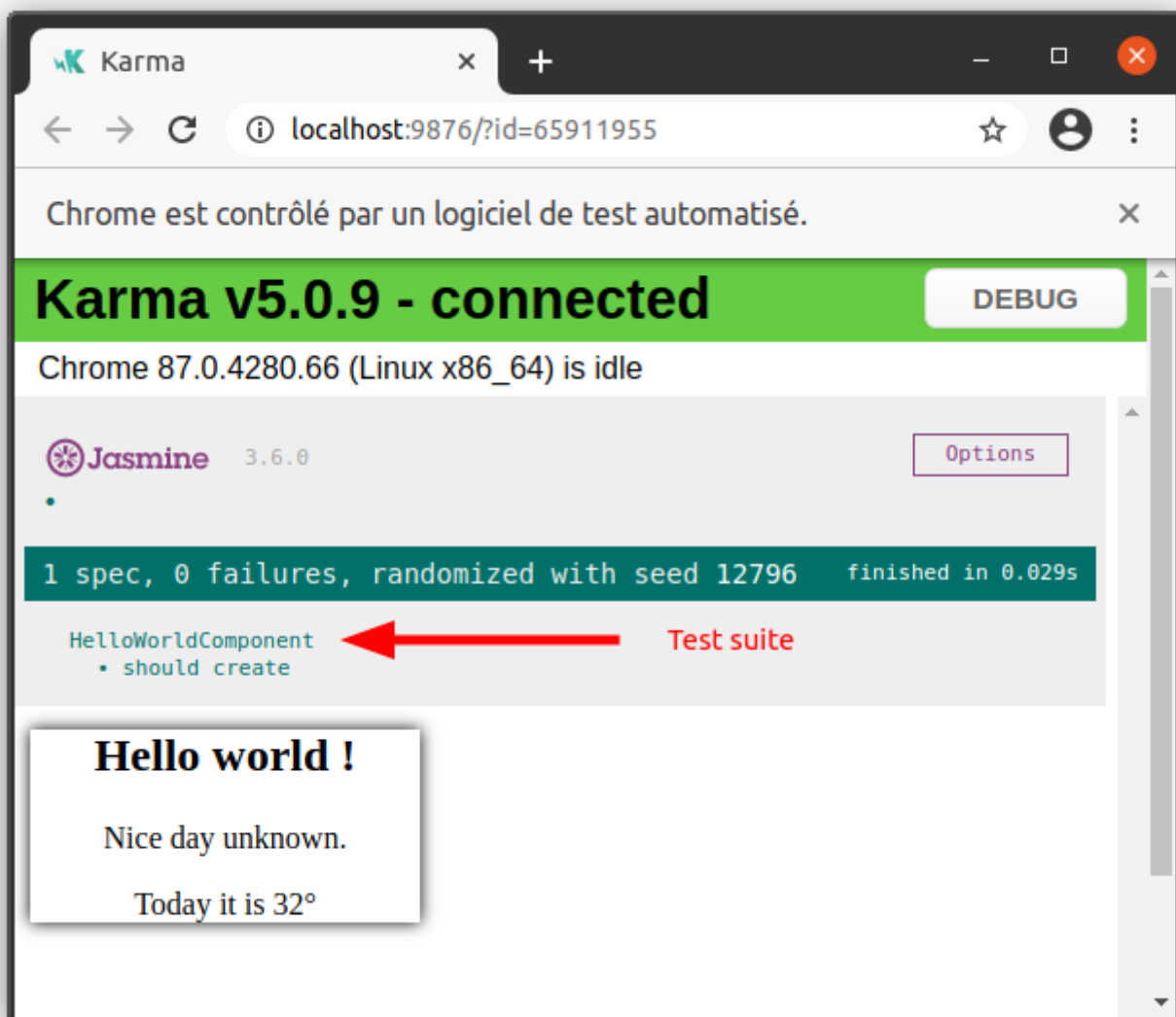
Pour lancer une suite donnée de tests, la syntaxe est :

```
ng test --include='**/someFolder/*.spec.ts'
```


Listing 5. `ng test --include='**/hello-world/*.spec.ts'`

```
Karma v5.0.9 server started at http://0.0.0.0:9876/
:INFO [launcher]: Starting browser Chrome
:WARN [karma]: No captured browser, open http://localhost:9876/
:INFO [Chrome 87.0.4280.66 (Linux x86_64)]: Connected on socket
Executed 1 of 1 SUCCESS (0.13 secs / 0.057 secs)
TOTAL: 1 SUCCESS
TOTAL: 1 SUCCESS
```

Voici le rapport de test, passé au vert, rendu par l'instance du navigateur support :



Sans surprise, le seul test inclus dans la suite (*should create*) est passé !



Il est temps de prendre une pause. Voici un peu de lecture qui vous permettra de découvrir quelques méthodes de tests `toBeGreaterThan`, `toEqual`... : https://jasmine.github.io/tutorials/your_first_suite

Ajout de tests spécifiques

Il est temps maintenant de vérifier si le composant se comporte comme attendu, c'est à dire s'il est fidèle à ses spécifications (d'où le suffixe `.spec.ts` du fichier regroupant les tests).

Dans un premier temps nous testerons la vue et sa mise à jour avec le modèle (propriété de la classe TS du composant), puis nous ferons évoluer le composant afin qu'il s'appuie sur un service donnant une température.

Test des valeurs par défaut

Vérification de l'état de l'instance gérant les données du modèle

Listing 6. ajout à la "test suite" `describe('HelloWorldComponent', ...`

```
[...]  
  
it('default name', () => {  
  expect(component.name).toBe("unknown");  
});
```

① : `"default name"` est le nom de la spec

② : Vérifie la valeur de la propriété `name` de l'objet référencé par `component`

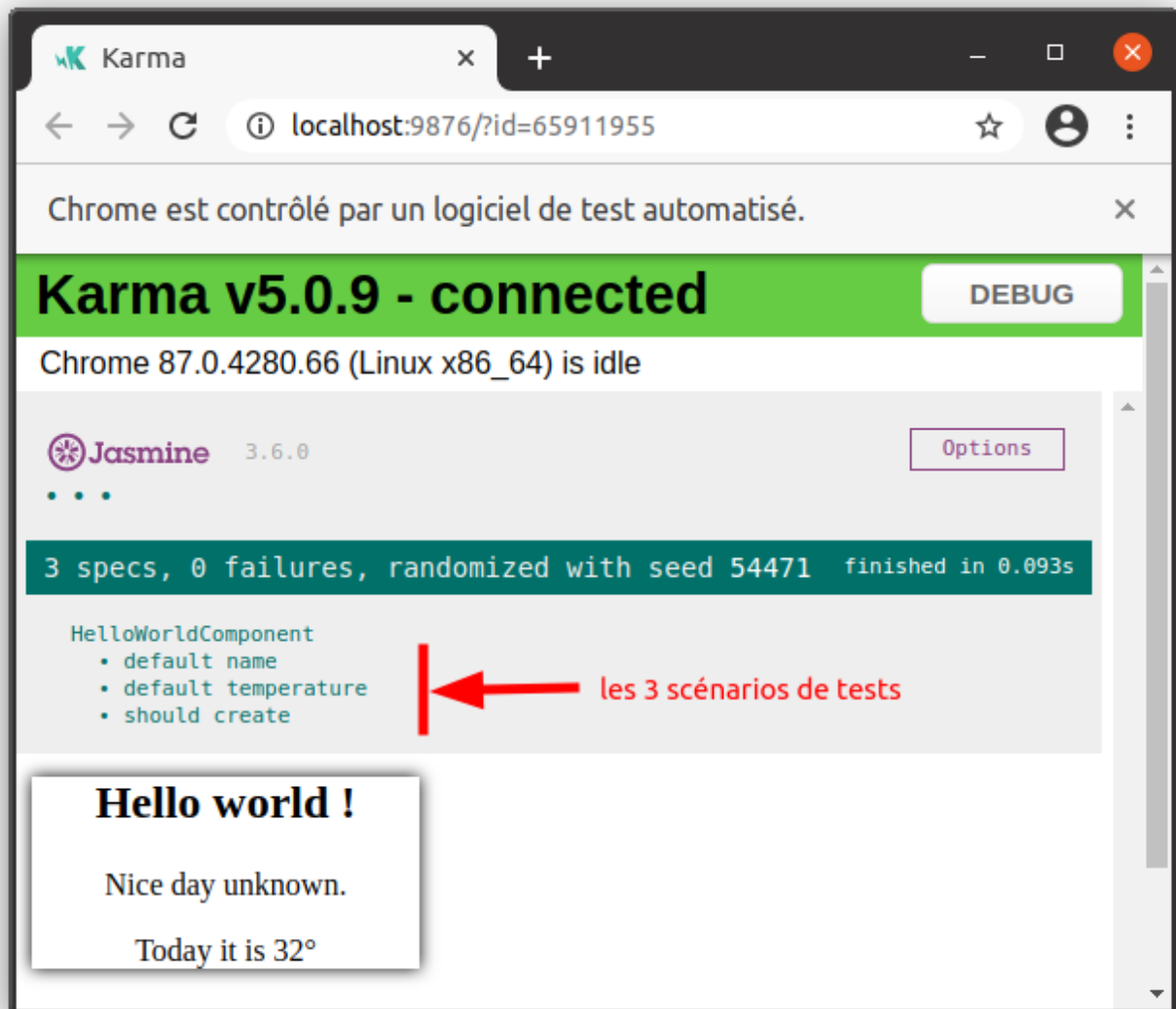
Nous ferons de meme avec la température

Listing 7. vérifier la température par défaut

```
[...]  
  
it('default temperature', () => {  
  expect(component.temperature.temp).toBe(32);  
});
```

① : La temperature étant représentée par un objet, nous accédons ici directement à sa propriété public `temp` (qui est en lecture seule)

Vérification (*karma* est normalement toujours actif, et relance les tests après chaque nouvelle compilation)



Nous pouvons vérifier si le composant réagit bien aux changements de valeur de certains de ses propriétés :

Listing 8. vérifier la possibilité de changer 'name'

```
[...]
it('new component.name', () => {
  component.name = "newName";
  expect(component.name).toBe("newName");
});
```

Test de réactivité avec la vue

Nous allons vérifier le lien de réactivité entre le modèle et la vue. Pour cela nous interrogeons la partie de DOM occupée par le composant.

Listing 9. it - view default view name

```
1 it('view default view name', () => {  
2   const rootElt = fixture.nativeElement; ①  
3   // console.log(rootElt);  
4   const firstP = rootElt.querySelector("div p:first-of-type").textContent; ②  
5   expect(firstP).toContain('unknown'); ③  
6 });
```

① : Obtenir le noeud racine du template du composant

② : C'est ici que l'on fera usage de **sélecteur CSS** pour atteindre les parties souhaitées. Dans le cas présent, on cherche à atteindre le premier `<p>` dans le seul `<div>` du rendu HTML du composant.

③ : Vérifie que la chaîne "unknown" est inclus dans les texte référencé par `firstP`.



Une bonne connaissance des possibilités des sélecteurs **CSS** s'impose ici. C'est une bonne raison de revisiter des ressources web sur ce sujet, et de s'améliorer ! voir par exemple [developer.mozilla Apprendre CSS/Selector](#) et [first-of-type selector](#)

Voici un autre test qui vérifie la réactivité de la vue face à un changement de son modèle (instance de la classe du composant).

Listing 10. it - view update name

```
1 it('view update name', () => {  
2   const rootElt = fixture.nativeElement;  
3   component.name = "newName";  
4   fixture.detectChanges();  
5   const firstP = rootElt.querySelector("div p:first-of-type").textContent;  
6   expect(firstP).toContain('newName');  
7 });
```

Exercice

- Concevoir une nouvelle spécification (un nouveau test unitaire) qui vérifie que la température présentée par le composant est bien celle attendue.

Evolution du composant

Recherche d'un service API de requête de température. Il en existe de nombreux, nécessitant la plupart du temps une clé d'accès, et un abonnement au service.

Pour les besoins de ce support, nous utiliserons le service proposé par [prevision-meteo.ch](#).

Testez par vous-même : [infos ville de Melun 77000 France](#)

Paramètre de l'application

Nous allons définir un service sous forme d'une classe qui aura la charge de nous fournir une

donnée de température.

Pour cela nous commençons par ajouter une classe qui contiendra des données globales, comme l'URL de l'API météo.

```
ng generate class GlobalConstants
```

Listing 11. global-constant.ts

```
export class GlobalConstants {  
  static readonly meteoUrlAPI : string = "https://www.prevision-  
meteo.ch/services/json/";  
}
```

Il est logique que le service météo nous retourne une instance de `Temperature`. Ce type mérite donc d'être déclaré dans un fichier à part. Nous le placerons dans un dossier nommé `model` :

```
ng generate interface Temperature --path=src/app/model
```

Listing 12. temperature.ts

```
export interface Temperature {  
  readonly temp : number  
}
```

Parallèlement, nous supprimons la définition de `Temperature` dans `hello-world.component.ts`, et déclarons à la place une dépendance, plus quelques aménagements **temporaires** expliqués ci-après. Le but de ces aménagements temporaires est de faire passer les tests unitaires actuels :

```

1 import { Component, Input, OnInit } from '@angular/core';
2 import { Temperature } from '../model/temperature';
3 import { MeteoService } from '../service/meteo/meteo.service';
4
5 @Component({
6   selector: 'app-hello-world',
7   templateUrl: './hello-world.component.html',
8   styleUrls: ['./hello-world.component.scss']
9 })
10 export class HelloWorldComponent implements OnInit {
11
12   @Input() name: string = "unknown";
13
14   public get temperature(): Temperature {           ①
15     return this.meteoService.getTemperatureFromCity("MeLun"); ②
16   }
17
18   constructor(private meteoService: MeteoService) { } ③
19
20   ngOnInit(): void {}
21
22 }

```

- ① un *getter* est un accesseur automatique (c'est du js). Chaque accès en lecture à *temperature* (sans parenthèses en fin) passera par le code de ce *get*.
- ② ici, nous appelons le service de *meteoService*.
- ③ déclaration d'un service qui sera injecté automatiquement par Angular, en tant qu'attribut privé (*property*) de la classe.

Début de creation du service météo (première version)

Nous placerons ce service dans un dossier dédié (via `cli ng`) :

```
ng generate service Meteo --flat=false --path=src/app/service
```

Listing 14. *src/app/service/meteo/meteo.service.ts*

```
1 import { Injectable } from '@angular/core';
2 import { Temperature } from 'src/app/model/temperature';
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class MeteoService {
8
9   getTemperatureFromCity(city : string) : Temperature {
10     return {temp : 32};
11   }
12
13   constructor() { }
14 }
```

À ce niveau d'avancement, les précédents tests unitaires devraient tous passer.



Nous venons de réaliser un **refactoring**. C'est très souvent le pris à payer pour favoriser les évolutions à venir.

Update de l'interface Temperature

Dans l'interface `Temperature` nous ajoutons le nom de la ville concernée.

Listing 15. *src/app/model/temperature.ts*

```
1 export interface Temperature {
2   readonly temp : number
3   readonly city : string
4 }
```

et bien entendu, le composant présente cette valeur à l'utilisateur :

Listing 16. src/app/hello-world/hello-world.component.html

```
1 <style>
2 .box {
3   box-shadow: inset 0 0 1em white, 0 0 .5em black;
4   text-align: center;
5   width: 200px;
6 }
7 </style>
8
9 <div class="box">
10   <h2 class="title">Hello world !</h2>
11   <p>Nice day {{name}}.</p>
12   <p>Today at {{temperature.city}} it is {{temperature.temp}}°</p>
13 </div>
```

Nous ne manquerons pas de mettre à jour la structure de l'objet retourné par la méthode du service météo :

Listing 17. src/app/service/meteo/meteo.service.ts

```
1 [...]
2
3 getTemperatureFromCity(city : string) : Temperature {
4   return {city: "Melun", temp : 32};
5 }
```

À ce moment des modifications, la suite de tests `HelloWorldComponent` devrait passer.

Nous allons maintenant demander au service d'effectuer une requête http à la demande d'une température.

Listing 18. *src/app/service/meteo/meteo.service.ts*

```
1 import { Injectable } from '@angular/core';
2 import { GlobalConstants } from 'src/app/global-constants'
3 import { Observable, of } from 'rxjs';
4 import { HttpClient } from '@angular/common/http';
5
6 @Injectable({
7   providedIn: 'root'
8 })
9 export class MeteoService {
10
11   constructor(private httpClient: HttpClient) { } ①
12
13   getTemperatureFromCity(city: string): Observable<any> {
14     return this.httpClient.get(this.getBaseUrl() + "/" + city); ②
15   }
16
17   getBaseUrl() : string { ③
18     return GlobalConstants.meteoUrlAPI;
19   }
20 }
```

- ① Définition d'une dépendance à une instance de `HttpClient`.
- ② Lancement de la requête HTTP, qui retourne un objet de type `Observable`.
- ③ Afin de connaître, dans les tests, l'URL utilisée.

Nous voici avec un service qui dépend d'un autre service avec appel asynchrone... Pas simple à tester.

Heureusement, Angular vient avec des bibliothèques de classes dédiées à ce problème.

Listing 19. *src/app/service/meteo/meteo.service.spec.ts*

```
1 import { HttpTestingController, HttpClientTestingModule } from
  '@angular/common/http/testing';
2 import { TestBed } from '@angular/core/testing';
3 import { MeteoService } from './meteo.service';
4
5 // voir : https://angular.io/guide/http#testing-http-requests
6
7 describe('MeteoService', () => {
8   let service: MeteoService;
9   let mockHttpTestingController: HttpTestingController; ①
10
11   beforeEach(() => {
12     TestBed.configureTestingModule({
13       imports: [HttpClientTestingModule],
14       providers: [MeteoService]
15     });
16   });
17
18   // ...
19
20   // ...
21
22   // ...
23
24   // ...
25
26   // ...
27
28   // ...
29
30   // ...
31
32   // ...
33
34   // ...
35
36   // ...
37
38   // ...
39
40   // ...
41
42   // ...
43
44   // ...
45
46   // ...
47
48   // ...
49
50   // ...
51
52   // ...
53
54   // ...
55
56   // ...
57
58   // ...
59
60   // ...
61
62   // ...
63
64   // ...
65
66   // ...
67
68   // ...
69
70   // ...
71
72   // ...
73
74   // ...
75
76   // ...
77
78   // ...
79
80   // ...
81
82   // ...
83
84   // ...
85
86   // ...
87
88   // ...
89
90   // ...
91
92   // ...
93
94   // ...
95
96   // ...
97
98   // ...
99
100  // ...
101
102  // ...
103
104  // ...
105
106  // ...
107
108  // ...
109
110  // ...
111
112  // ...
113
114  // ...
115
116  // ...
117
118  // ...
119
120  // ...
121
122  // ...
123
124  // ...
125
126  // ...
127
128  // ...
129
130  // ...
131
132  // ...
133
134  // ...
135
136  // ...
137
138  // ...
139
140  // ...
141
142  // ...
143
144  // ...
145
146  // ...
147
148  // ...
149
150  // ...
151
152  // ...
153
154  // ...
155
156  // ...
157
158  // ...
159
160  // ...
161
162  // ...
163
164  // ...
165
166  // ...
167
168  // ...
169
170  // ...
171
172  // ...
173
174  // ...
175
176  // ...
177
178  // ...
179
180  // ...
181
182  // ...
183
184  // ...
185
186  // ...
187
188  // ...
189
190  // ...
191
192  // ...
193
194  // ...
195
196  // ...
197
198  // ...
199
200  // ...
201
202  // ...
203
204  // ...
205
206  // ...
207
208  // ...
209
210  // ...
211
212  // ...
213
214  // ...
215
216  // ...
217
218  // ...
219
220  // ...
221
222  // ...
223
224  // ...
225
226  // ...
227
228  // ...
229
230  // ...
231
232  // ...
233
234  // ...
235
236  // ...
237
238  // ...
239
240  // ...
241
242  // ...
243
244  // ...
245
246  // ...
247
248  // ...
249
250  // ...
251
252  // ...
253
254  // ...
255
256  // ...
257
258  // ...
259
260  // ...
261
262  // ...
263
264  // ...
265
266  // ...
267
268  // ...
269
270  // ...
271
272  // ...
273
274  // ...
275
276  // ...
277
278  // ...
279
280  // ...
281
282  // ...
283
284  // ...
285
286  // ...
287
288  // ...
289
290  // ...
291
292  // ...
293
294  // ...
295
296  // ...
297
298  // ...
299
300  // ...
301
302  // ...
303
304  // ...
305
306  // ...
307
308  // ...
309
310  // ...
311
312  // ...
313
314  // ...
315
316  // ...
317
318  // ...
319
320  // ...
321
322  // ...
323
324  // ...
325
326  // ...
327
328  // ...
329
330  // ...
331
332  // ...
333
334  // ...
335
336  // ...
337
338  // ...
339
340  // ...
341
342  // ...
343
344  // ...
345
346  // ...
347
348  // ...
349
350  // ...
351
352  // ...
353
354  // ...
355
356  // ...
357
358  // ...
359
360  // ...
361
362  // ...
363
364  // ...
365
366  // ...
367
368  // ...
369
370  // ...
371
372  // ...
373
374  // ...
375
376  // ...
377
378  // ...
379
380  // ...
381
382  // ...
383
384  // ...
385
386  // ...
387
388  // ...
389
390  // ...
391
392  // ...
393
394  // ...
395
396  // ...
397
398  // ...
399
400  // ...
401
402  // ...
403
404  // ...
405
406  // ...
407
408  // ...
409
410  // ...
411
412  // ...
413
414  // ...
415
416  // ...
417
418  // ...
419
420  // ...
421
422  // ...
423
424  // ...
425
426  // ...
427
428  // ...
429
430  // ...
431
432  // ...
433
434  // ...
435
436  // ...
437
438  // ...
439
440  // ...
441
442  // ...
443
444  // ...
445
446  // ...
447
448  // ...
449
450  // ...
451
452  // ...
453
454  // ...
455
456  // ...
457
458  // ...
459
460  // ...
461
462  // ...
463
464  // ...
465
466  // ...
467
468  // ...
469
470  // ...
471
472  // ...
473
474  // ...
475
476  // ...
477
478  // ...
479
480  // ...
481
482  // ...
483
484  // ...
485
486  // ...
487
488  // ...
489
490  // ...
491
492  // ...
493
494  // ...
495
496  // ...
497
498  // ...
499
500  // ...
501
502  // ...
503
504  // ...
505
506  // ...
507
508  // ...
509
510  // ...
511
512  // ...
513
514  // ...
515
516  // ...
517
518  // ...
519
520  // ...
521
522  // ...
523
524  // ...
525
526  // ...
527
528  // ...
529
530  // ...
531
532  // ...
533
534  // ...
535
536  // ...
537
538  // ...
539
540  // ...
541
542  // ...
543
544  // ...
545
546  // ...
547
548  // ...
549
550  // ...
551
552  // ...
553
554  // ...
555
556  // ...
557
558  // ...
559
560  // ...
561
562  // ...
563
564  // ...
565
566  // ...
567
568  // ...
569
570  // ...
571
572  // ...
573
574  // ...
575
576  // ...
577
578  // ...
579
580  // ...
581
582  // ...
583
584  // ...
585
586  // ...
587
588  // ...
589
590  // ...
591
592  // ...
593
594  // ...
595
596  // ...
597
598  // ...
599
600  // ...
601
602  // ...
603
604  // ...
605
606  // ...
607
608  // ...
609
610  // ...
611
612  // ...
613
614  // ...
615
616  // ...
617
618  // ...
619
620  // ...
621
622  // ...
623
624  // ...
625
626  // ...
627
628  // ...
629
630  // ...
631
632  // ...
633
634  // ...
635
636  // ...
637
638  // ...
639
640  // ...
641
642  // ...
643
644  // ...
645
646  // ...
647
648  // ...
649
650  // ...
651
652  // ...
653
654  // ...
655
656  // ...
657
658  // ...
659
660  // ...
661
662  // ...
663
664  // ...
665
666  // ...
667
668  // ...
669
670  // ...
671
672  // ...
673
674  // ...
675
676  // ...
677
678  // ...
679
680  // ...
681
682  // ...
683
684  // ...
685
686  // ...
687
688  // ...
689
690  // ...
691
692  // ...
693
694  // ...
695
696  // ...
697
698  // ...
699
700  // ...
701
702  // ...
703
704  // ...
705
706  // ...
707
708  // ...
709
710  // ...
711
712  // ...
713
714  // ...
715
716  // ...
717
718  // ...
719
720  // ...
721
722  // ...
723
724  // ...
725
726  // ...
727
728  // ...
729
730  // ...
731
732  // ...
733
734  // ...
735
736  // ...
737
738  // ...
739
740  // ...
741
742  // ...
743
744  // ...
745
746  // ...
747
748  // ...
749
750  // ...
751
752  // ...
753
754  // ...
755
756  // ...
757
758  // ...
759
760  // ...
761
762  // ...
763
764  // ...
765
766  // ...
767
768  // ...
769
770  // ...
771
772  // ...
773
774  // ...
775
776  // ...
777
778  // ...
779
780  // ...
781
782  // ...
783
784  // ...
785
786  // ...
787
788  // ...
789
790  // ...
791
792  // ...
793
794  // ...
795
796  // ...
797
798  // ...
799
800  // ...
801
802  // ...
803
804  // ...
805
806  // ...
807
808  // ...
809
810  // ...
811
812  // ...
813
814  // ...
815
816  // ...
817
818  // ...
819
820  // ...
821
822  // ...
823
824  // ...
825
826  // ...
827
828  // ...
829
830  // ...
831
832  // ...
833
834  // ...
835
836  // ...
837
838  // ...
839
840  // ...
841
842  // ...
843
844  // ...
845
846  // ...
847
848  // ...
849
850  // ...
851
852  // ...
853
854  // ...
855
856  // ...
857
858  // ...
859
860  // ...
861
862  // ...
863
864  // ...
865
866  // ...
867
868  // ...
869
870  // ...
871
872  // ...
873
874  // ...
875
876  // ...
877
878  // ...
879
880  // ...
881
882  // ...
883
884  // ...
885
886  // ...
887
888  // ...
889
890  // ...
891
892  // ...
893
894  // ...
895
896  // ...
897
898  // ...
899
900  // ...
901
902  // ...
903
904  // ...
905
906  // ...
907
908  // ...
909
910  // ...
911
912  // ...
913
914  // ...
915
916  // ...
917
918  // ...
919
920  // ...
921
922  // ...
923
924  // ...
925
926  // ...
927
928  // ...
929
930  // ...
931
932  // ...
933
934  // ...
935
936  // ...
937
938  // ...
939
940  // ...
941
942  // ...
943
944  // ...
945
946  // ...
947
948  // ...
949
950  // ...
951
952  // ...
953
954  // ...
955
956  // ...
957
958  // ...
959
960  // ...
961
962  // ...
963
964  // ...
965
966  // ...
967
968  // ...
969
970  // ...
971
972  // ...
973
974  // ...
975
976  // ...
977
978  // ...
979
980  // ...
981
982  // ...
983
984  // ...
985
986  // ...
987
988  // ...
989
990  // ...
991
992  // ...
993
994  // ...
995
996  // ...
997
998  // ...
999
1000  // ...
1001
1002  // ...
1003
1004  // ...
1005
1006  // ...
1007
1008  // ...
1009
1010  // ...
1011
1012  // ...
1013
1014  // ...
1015
1016  // ...
1017
1018  // ...
1019
1020  // ...
1021
1022  // ...
1023
1024  // ...
1025
1026  // ...
1027
1028  // ...
1029
1030  // ...
1031
1032  // ...
1033
1034  // ...
1035
1036  // ...
1037
1038  // ...
1039
1040  // ...
1041
1042  // ...
1043
1044  // ...
1045
1046  // ...
1047
1048  // ...
1049
1050  // ...
1051
1052  // ...
1053
1054  // ...
1055
1056  // ...
1057
1058  // ...
1059
1060  // ...
1061
1062  // ...
1063
1064  // ...
1065
1066  // ...
1067
1068  // ...
1069
1070  // ...
1071
1072  // ...
1073
1074  // ...
1075
1076  // ...
1077
1078  // ...
1079
1080  // ...
1081
1082  // ...
1083
1084  // ...
1085
1086  // ...
1087
1088  // ...
1089
1090  // ...
1091
1092  // ...
1093
1094  // ...
1095
1096  // ...
1097
1098  // ...
1099
1100  // ...
1101
1102  // ...
1103
1104  // ...
1105
1106  // ...
1107
1108  // ...
1109
1110  // ...
1111
1112  // ...
1113
1114  // ...
1115
1116  // ...
1117
1118  // ...
1119
1120  // ...
1121
1122  // ...
1123
1124  // ...
1125
1126  // ...
1127
1128  // ...
1129
1130  // ...
1131
1132  // ...
1133
1134  // ...
1135
1136  // ...
1137
1138  // ...
1139
1140  // ...
1141
1142  // ...
1143
1144  // ...
1145
1146  // ...
1147
1148  // ...
1149
1150  // ...
1151
1152  // ...
1153
1154  // ...
1155
1156  // ...
1157
1158  // ...
1159
1160  // ...
1161
1162  // ...
1163
1164  // ...
1165
1166  // ...
1167
1168  // ...
1169
1170  // ...
1171
1172  // ...
1173
1174  // ...
1175
1176  // ...
1177
1178  // ...
1179
1180  // ...
1181
1182  // ...
1183
1184  // ...
1185
1186  // ...
1187
1188  // ...
1189
1190  // ...
1191
1192  // ...
1193
1194  // ...
1195
1196  // ...
1197
1198  // ...
1199
1200  // ...
1201
1202  // ...
1203
1204  // ...
1205
1206  // ...
1207
1208  // ...
1209
1210  // ...
1211
1212  // ...
1213
1214  // ...
1215
1216  // ...
1217
1218  // ...
1219
1220  // ...
1221
1222  // ...
1223
1224  // ...
1225
1226  // ...
1227
1228  // ...
1229
1230  // ...
1231
1232  // ...
1233
1234  // ...
1235
1236  // ...
1237
1238  // ...
1239
1240  // ...
1241
1242  // ...
1243
1244  // ...
1245
1246  // ...
1247
1248  // ...
1249
1250  // ...
1251
1252  // ...
1253
1254  // ...
1255
1256  // ...
1257
1258  // ...
1259
1260  // ...
1261
1262  // ...
1263
1264  // ...
1265
1266  // ...
1267
1268  // ...
1269
1270  // ...
1271
1272  // ...
1273
1274  // ...
1275
1276  // ...
1277
1278  // ...
1279
1280  // ...
1281
1282  // ...
1283
1284  // ...
1285
1286  // ...
1287
1288  // ...
1289
1290  // ...
1291
1292  // ...
1293
1294  // ...
1295
1296  // ...
1297
1298  // ...
1299
1300  // ...
1301
1302  // ...
1303
1304  // ...
1305
1306  // ...
1307
1308  // ...
1309
1310  // ...
1311
1312  // ...
1313
1314  // ...
1315
1316  // ...
1317
1318  // ...
1319
1320  // ...
1321
1322  // ...
1323
1324  // ...
1325
1326  // ...
1327
1328  // ...
1329
1330  // ...
1331
1332  // ...
1333
1334  // ...
1335
1336  // ...
1337
1338  // ...
1339
1340  // ...
1341
1342  // ...
1343
1344  // ...
1345
1346  // ...
1347
1348  // ...
1349
1350  // ...
1351
1352  // ...
1353
1354  // ...
1355
1356  // ...
1357
1358  // ...
1359
1360  // ...
1361
1362  // ...
1363
1364  // ...
1365
1366  // ...
1367
1368  // ...
1369
1370  // ...
1371
1372  // ...
1373
1374  // ...
1375
1376  // ...
1377
1378  // ...
1379
1380  // ...
1381
1382  // ...
1383
1384  // ...
1385
1386  // ...
1387
1388  // ...
1389
1390  // ...
1391
1392  // ...
1393
1394  // ...
1395
1396  // ...
1397
1398  // ...
1399
1400  // ...
1401
1402  // ...
1403
1404  // ...
1405
1406  // ...
1407
1408  // ...
1409
1410  // ...
1411
1412  // ...
1413
1414  // ...
1415
1416  // ...
1417
1418  // ...
1419
1420  // ...
1421
1422  // ...
1423
1424  // ...
1425
1426  // ...
1427
1428  // ...
1429
1430  // ...
1431
1432  // ...
1433
1434  // ...
1435
1436  // ...
1437
1438  // ...
1439
1440  // ...
1441
1442  // ...
1443
1444  // ...
1445
1446  // ...
1447
1448  // ...
1449
1450  // ...
1451
1452  // ...
1453
1454  // ...
1455
1456  // ...
1457
1458  // ...
1459
1460  // ...
1461
1462  // ...
1463
1464  // ...
1465
1466  // ...
1467
1468  // ...
1469
1470  // ...
1471
1472  // ...
1473
1474  // ...
1475
1476  // ...
1477
1478  // ...
1479
1480  // ...
1481
1482  // ...
1483
1484  // ...
1485
1486  // ...
1487
1488  // ...
1489
1490  // ...
1491
1492  // ...
1493
1494  // ...
1495
1496  // ...
1497
1498  // ...
1499
1500  // ...
1501
1502  // ...
1503
1504  // ...
1505
1506  // ...
1507
1508  // ...
1509
1510  // ...
1511
1512  // ...
1513
1514  // ...
1515
1516  // ...
1517
1518  // ...
1519
1520  // ...
1521
1522  // ...
1523
1524  // ...
1525
1526  // ...
1527
1528  // ...
1529
1530  // ...
1531
1532  // ...
1533
1534  // ...
1535
1536  // ...
1537
1538  // ...
1539
1540  // ...
1541
1542  // ...
1543
1544  // ...
1545
1546  // ...
1547
1548  // ...
1549
1550  // ...
1551
1552  // ...
1553
1554  // ...
1555
1556  // ...
1557
1558  // ...
1559
1560  // ...
1561
1562  // ...
1563
1564  // ...
1565
1566  // ...
1567
1568  // ...
1569
1570  // ...
1571
1572  // ...
1573
1574  // ...
1575
1576  // ...
1577
1578  // ...
1579
1580  // ...
1581
1582  // ...
1583
1584  // ...
1585
1586  // ...
1587
1588  // ...
1589
1590  // ...
1591
1592  // ...
1593
1594  // ...
1595
1596  // ...
1597
1598  // ...
1599
1600  // ...
1601
1602  // ...
1603
1604  // ...
1605
1606  // ...
1607
1608  // ...
1609
1610  // ...
1611
1612  // ...
1613
1614  // ...
1615
1616  // ...
1617
1618  // ...
1619
1620  // ...
1621
1622  // ...
1623
1624  // ...
1625
1626  // ...
1627
1628  // ...
1629
1630  // ...
1631
1632  // ...
1633
1634  // ...
1635
1636  // ...
1637
1638  // ...
1639
1640  // ...
1641
1642  // ...
1643
1644  // ...
1645
1646  // ...
1647
1648  // ...
1649
1650  // ...
1651
1652  // ...
1653
1654  // ...
1655
1656  // ...
1657
1658  // ...
1659
1660  // ...
1661
1662  // ...
1663
1664  // ...
1665
1666  // ...
1667
1668  // ...
1669
1670  // ...
1671
1672  // ...
1673
1674  // ...
1675
1676  // ...
1677
1678  // ...
1679
1680  // ...
1681
1682  // ...
1683
1684  // ...
1685
1686  // ...
1687
1688  // ...
1689
1690  // ...
1691
1692  // ...
1693
1694  // ...
1695
1696  // ...
1697
1698  // ...
1699
1700  // ...
1701
1702  // ...
1703
1704  // ...
1705
1706  // ...
1707
1708  // ...
1709
1710  // ...
1711
1712  // ...
1713
1714  // ...
1715
1716  // ...
1717
1718  // ...
1719
1720  // ...
1721
1722  // ...
1723
1724  // ...
1725
1726  // ...
1727
1728  // ...
1729
1730  // ...
1731
1732  // ...
1733
1734  // ...
1735
1736  // ...
1737
1738  // ...
1739
1740  // ...
1741
1742  // ...
1743
1744  // ...
1745
1746  // ...
1747
1748  // ...
1749
1750  // ...
1751
1752  // ...
1753
1754  // ...
1755
1756  // ...
1757
1758  // ...
1759
1760  // ...
1761
1762  // ...
1763
1764  // ...
1765
1766  // ...
1767
1768  // ...
1769
1770  // ...
1771
1772  // ...
1773
1774  // ...
1775
1776  // ...
1777
1778  // ...
1779
1780  // ...
1781
1782  // ...
1783
1784  // ...
1785
1786  // ...
1787
1788  // ...
1789
1790  // ...
1791
1792  // ...
1793
1794  // ...
1795
1796  // ...
1797
1798  // ...
1799
1800  // ...
1801
1802  // ...
1803
1804  // ...
1805
1806  // ...
1807
1808  // ...
1809
1810  // ...
1811
1812  // ...
1813
1814  // ...
1815
1816  // ...
1817
1818  // ...
1819
1820  // ...
1821
1822  // ...
1823
1824  // ...
1825
1826  // ...
1827
1828  // ...
1829
1830  // ...
1831
1832  // ...
1833
1834  // ...
1835
1836  // ...
1837
1838  // ...
1839
1840  // ...
1841
1842  // ...
1843
1844  // ...
1845
1846  // ...
1847
1848  // ...
1849
1850  // ...
1851
1852  // ...
1853
1854  // ...
1855
1856  // ...
1857
1858  // ...
1859
1860  // ...
1861
1862  // ...
1863
1864  // ...
1865
1866  // ...
1867
1868  // ...
1869
1870  // ...
1871
1872  // ...
1873
1874  // ...
1875
1876  // ...
1877
1878  // ...
1879
1880  // ...
1881
1882  // ...
1883
1884  // ...
1885
1886  // ...
1887
1888  // ...
1889
1890  // ...
1891
1892  // ...
1893
1894  // ...
1895
1896  // ...
1897
1898  // ...
1899
1900  // ...
1901
1902  // ...
1903
1904  // ...
1905
1906  // ...
1907
1908  // ...
1909
1910  // ...
1911
1912  // ...
1913
1914  // ...
1915
1916  // ...
1917
1918  // ...
1919
1920  // ...
1921
1922  // ...
1923
1924  // ...
1925
1926  // ...
1927
1928  // ...
1929
1930  // ...
1931
1932  // ...
1933
1934  // ...
1935
1936  // ...
1937
1938  // ...
1939
1940  // ...
1941
1942  // ...
1943
1944  // ...
1945
1946  // ...
1947
1948  // ...
1949
1950  // ...
1951
1952  // ...
1953
1954  // ...
1955
1956  // ...
1957
1958  // ...
1959
1960  // ...
1961
1962  // ...
1963
1964  // ...
1965
1966  // ...
1967
1968  // ...
1969
1970  // ...
1971
1972  // ...
1973
1974  // ...
1975
197
```

```

16  service = TestBed.inject(MeteoService);
17  mockHttpTestingController = TestBed.inject(HttpTestingController); ②
18  });
19
20  it('should be created', () => {
21      expect(service).toBeTruthy(); ③
22  });
23
24  it('city known - Melun with temperature', () => {
25      let testUrl = service.getBaseUrl() + "/melun"; ④
26
27      service.getTemperatureFromCity("melun").subscribe(data => {
28          console.log("TEST RECEIVE TEMP : " + data?.current_condition?.temp); ⑤
29          expect(data?.current_condition?.temp).toBeGreaterThan(10);
30          expect(data?.city_info?.name).toEqual("Melun");
31          expect(data?.city_info?.country).toEqual("France");
32      });
33
34
35      const req = mockHttpTestingController.expectOne(testUrl);
36      expect(req.request.method).toEqual('GET');
37
38      let mockedResponse = { ⑥
39          city_info: {
40              name: "Melun",
41              country: "France"
42          },
43          current_condition: {
44              temp: 13
45          }
46      };
47
48      req.flush(mockedResponse); ⑦
49
50      mockHttpTestingController.verify(); ⑧
51  });
52
53  });

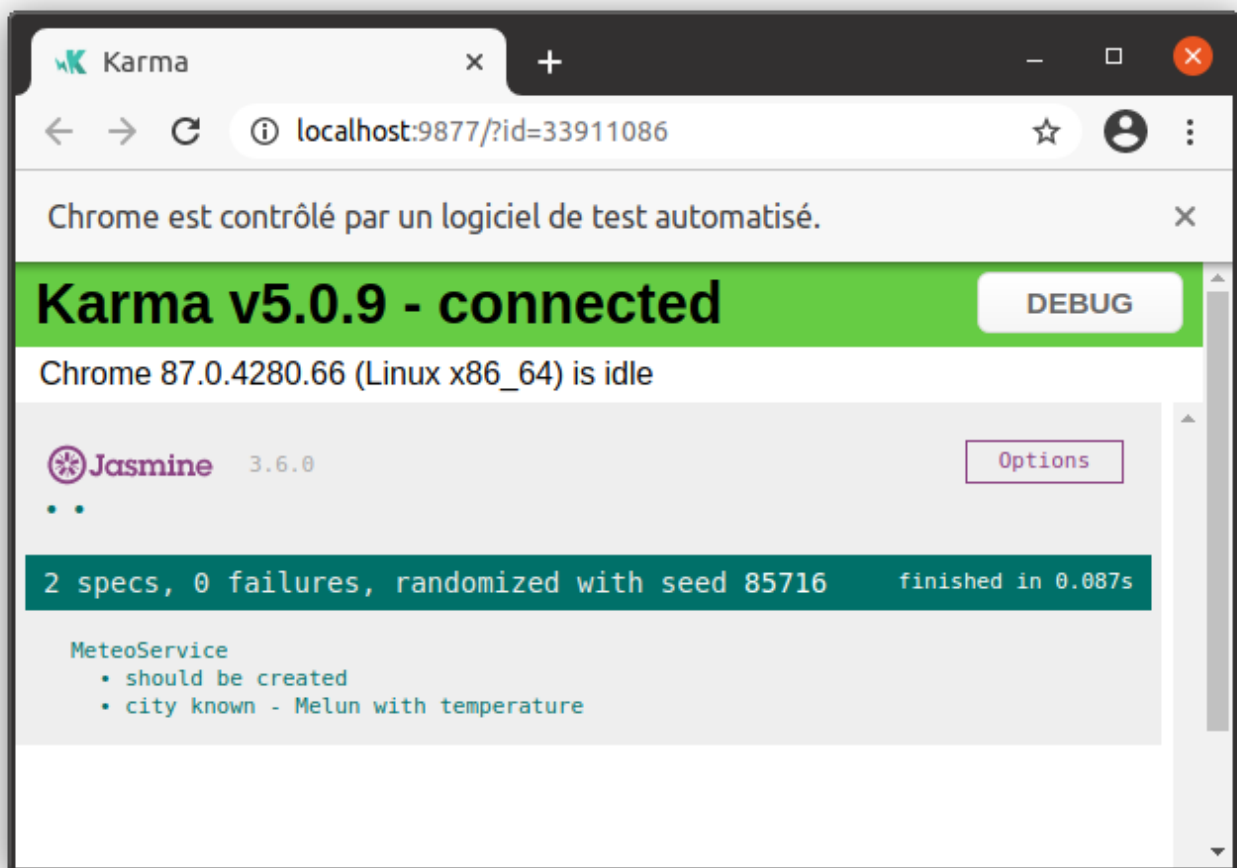
```

- ① Déclaration d'une référence à une instance de mock http
- ② Demande d'injection et récupération de l'instance
- ③ Le service existe t'il ?
- ④ Le url utilisées par le composant et le mock http doivent être identiques
- ⑤ Lorsque que la requête reçoit une réponse, les *souscripteurs* (les observateurs) en sont avisés. La donnée reçue est donnée comme argument (nommée **data**) de la fonction callback anonyme passée à la méthode `subscribe` : on peut donc ici coder les tests avec `expect`. On remarquera l'usage `?.` qui est une façon sûre (*safe*) de tenter d'atteindre un symbol dans une structure que nous ne maîtrisons pas (rend `undefined` en cas d'échec). (voir <https://www.typescriptlang.org/docs/handbook/release-notes/typescript-3-7.html>)

- ⑥ Construction des données simulant la réponse
- ⑦ Lance la réponse, **ce qui déclenchera la méthode resolve de l'observable** (ses abonnés en seront avisés)
- ⑧ Vérifie qu'il n'y a pas de demandes en suspens.

Vérification

```
ng test --include='**/service/meteo/*.spec.ts'
```



Conclusion

Nous avons présenté comment mettre en oeuvre des tests unitaires avec Angular, et par là même, via une méthodologie basée sur le refactoring.

Exercices

- Modifier l'interface `Temperature` afin qu'elle détienne le nom du pays (voir le json reçu du service API). Bien entendu des modifications devront être apportées ici et là dans le code. Faire passer les tests.
- Modifier de nouveau l'interface `Temperature` afin de présenter la température **maximale** et la

minimale du jour.

- Défi : Donner la tendance de la météo (une analyse basée sur les données des jours à venir, données reçues avec la réponse)