



---

# Tests unitaires / end 2 end

Animé par Mazen Gharbi

# Tests Angular

- ▷ Angular est un framework orienté tests
- ▷ « La documentation la plus à jour reste vos tests »
- ▷ Tout est d'ores et déjà configuré pour lancer vos tests avec la CLI



# Karma & Jasmine

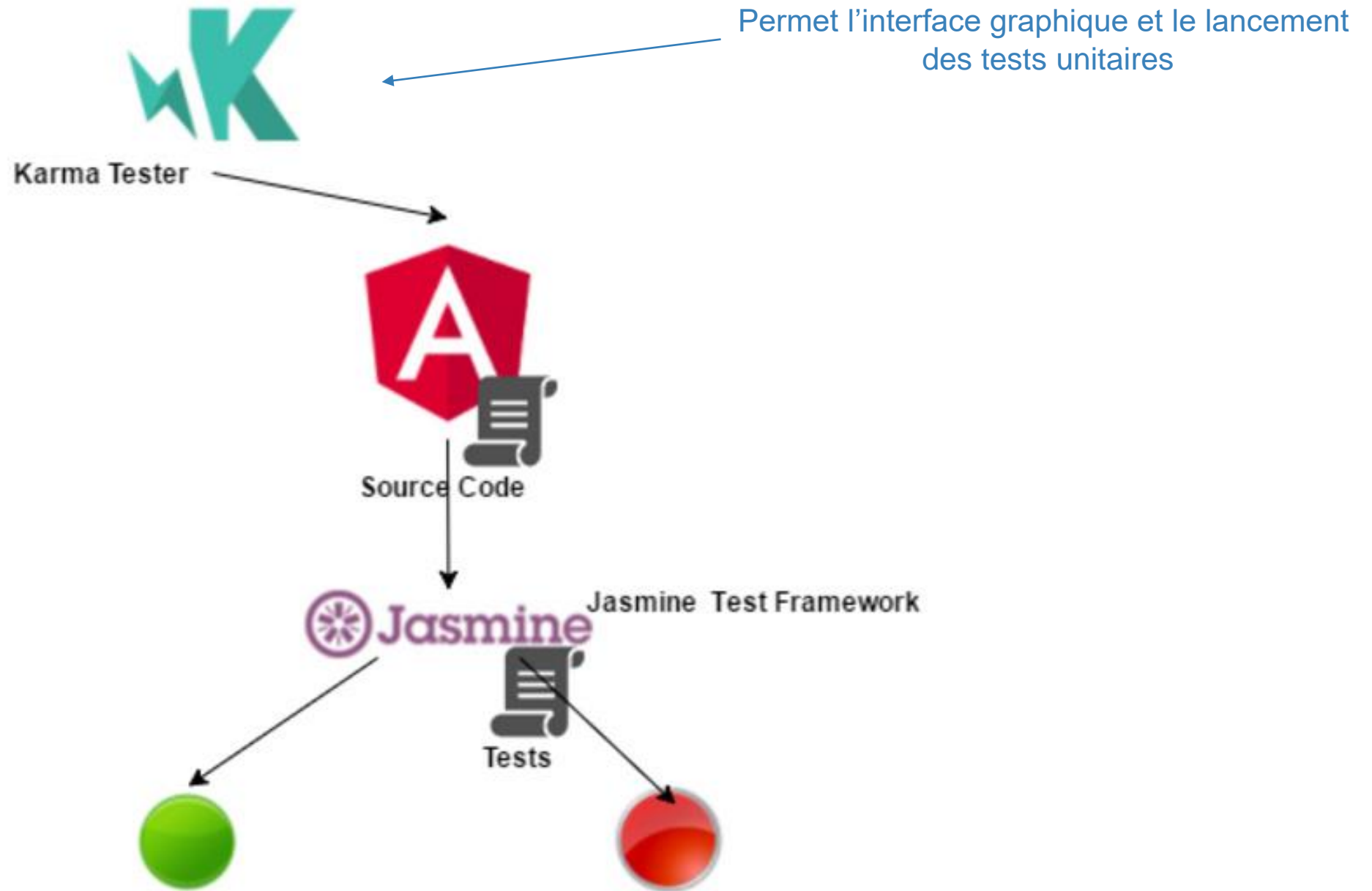
- ▷ Karma est un outil permettant d'exécuter les tests unitaires sur le navigateur de notre choix.



- ▷ Jasmine est le framework de « testing » intégré dans Angular. C'est le JUnit du JS.



- ▷ La convention est d'appeler chaque suite de test avec le même nom que le fichier à tester et le suffixe ".spec.ts".



# Tests unitaires - Angular


▷ Pour lancer vos tests :

```
> ng test
```

**Karma v4.1.0 - connected**

DEBUG

Chrome 77.0.3865 (Windows 10.0.0) is idle

 **Jasmine** 3.4.0  
...

Options

3 specs, 0 failures, randomized with seed 83202

finished in 0.418s

AppComponent  
• should create the app  
• should render title  
• should have as title 'demo-test'

# Line coverage

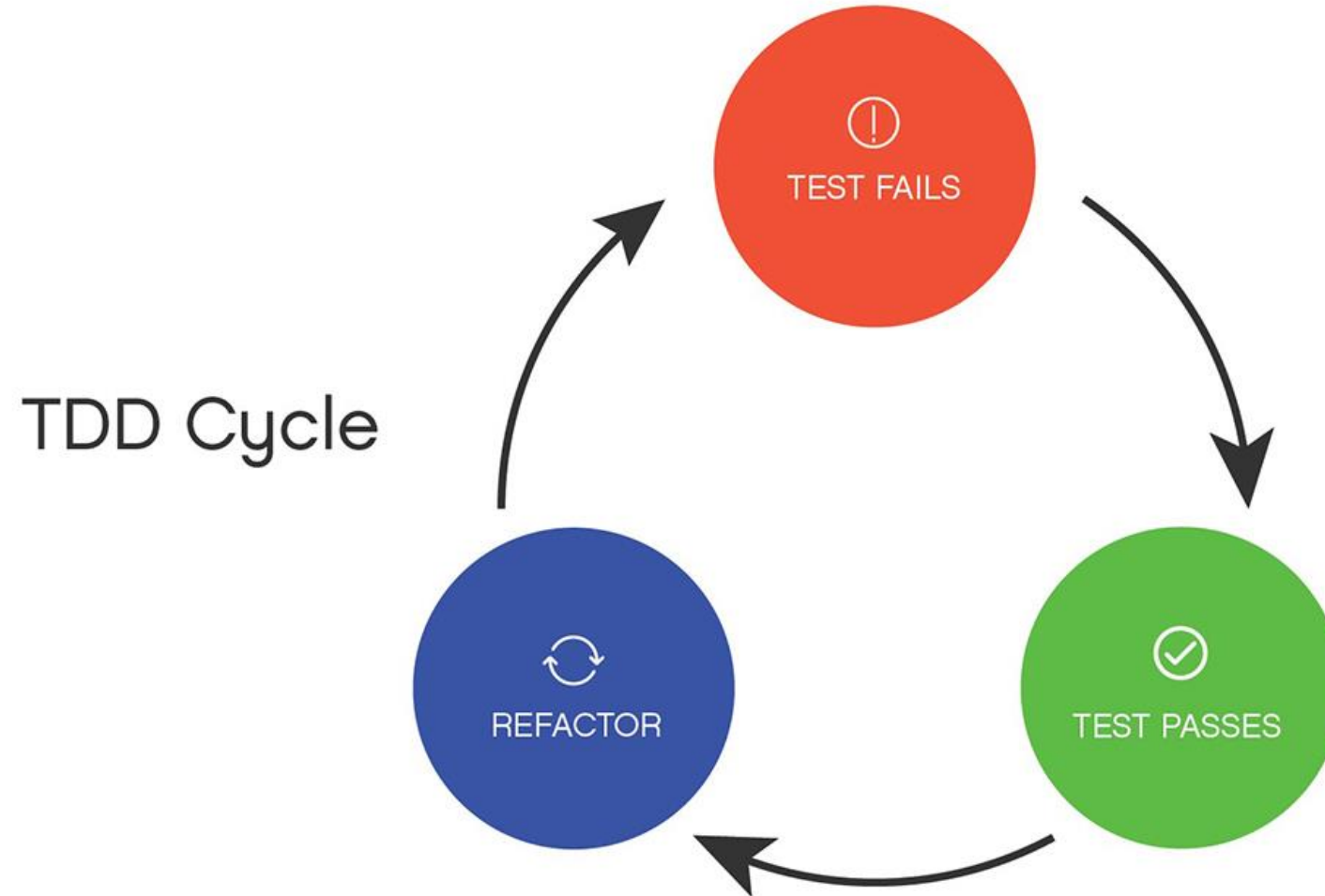
- ▷ Un petit argument à ajouter :

```
> ng test --code-coverage
```

↓ Istanbul

```
===== Coverage summary =====  
Statements : 100% ( 6/6 )  
Branches   : 100% ( 0/0 )  
Functions  : 100% ( 1/1 )  
Lines      : 100% ( 5/5 )  
=====
```

# Test Driven Development



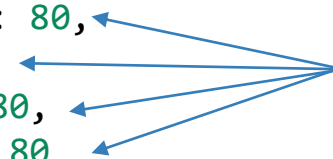
# Définissez vos attentes

- ▷ Vous pouvez définir un « line coverage » minimal avec la Angular CLI

```
=====
06 10 2019 20:27:28.725:ERROR [reporter.coverage-istanbul]: Coverage for statements (72.73%) does not meet global threshold (80%)
06 10 2019 20:27:28.726:ERROR [reporter.coverage-istanbul]: Coverage for lines (66.67%) does not meet global threshold (80%)
06 10 2019 20:27:28.727:ERROR [reporter.coverage-istanbul]: Coverage for functions (40%) does not meet global threshold (80%)
```

- ▷ Pour ce faire, rendez-vous dans le fichier « karma.conf.js », puis dans « coverageIstanbulReporter », ajoutez :

```
...
coverageIstanbulReporter: {
  reports: ['html', 'lcovonly'],
  fixWebpackSourcePaths: true,
  thresholds: {
    statements: 80,
    lines: 80,
    branches: 80,
    functions: 80
  }
}
...
```



Pourcentage



# Test d'une classe

```
import Calculator from './calculator.class';  
  
describe('Calculator', () => {  
  it('should say 4 when asked 2 * 2', () => {  
    const calculator = new Calculator();  
    expect(calculator.calculate('2 * 2')).toEqual(4);  
  });  
});
```

calculator.spec.ts

# Test d'une classe

calculator.class.ts

```
export default class Calculator {  
  calculate(formula: string) {  
    const [value1, value2] = formula.split('*');  
    return parseFloat(value1) * parseFloat(value2);  
  }  
}
```

# Service

calculator.service.spec.ts

```
describe('CalculatorService', () => {  
  beforeEach(() => TestBed.configureTestingModule({}));  
  
  it('should be created', () => {  
    const service: CalculatorService = TestBed.get(CalculatorService);  
    expect(service).toBeTruthy();  
  });  
  
  it('should say 4 when asked 2 * 2', () => {  
    const calculator: CalculatorService = TestBed.get(CalculatorService);  
    expect(calculator.calculate('2 * 2')).toEqual(4);  
  });  
});
```

<https://stackblitz.com/edit/angular-macademia-test-service>

# Composant

```
describe('Hello component', () => {
  beforeEach((done) => {
    TestBed
      .configureTestingModule({
        imports: [
          HelloModule
        ]
      })
      .compileComponents()
      .then(done);
  });

  it('should say hi', () => {
    let fixture = TestBed.createComponent(HelloComponent);

    fixture.componentInstance.name = 'John';
    fixture.detectChanges();

    expect(fixture.debugElement.nativeElement.innerText).toEqual('Hello John!');
  });
});
```

app.component.spec.ts

<https://stackblitz.com/edit/angular-macademia-test-component>

# Tests asynchrones

▷ Soit ce cas de figure :

user.service.ts

```
@Injectable({
  providedIn: 'root'
})
export class UserService {
  constructor() {}

  lateFeedback() {
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        resolve(true);
      }, 20000); // 20 SECONDES !
    });
  }
}
```

# Tests asynchrones

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


user.spec.service.ts

```
import { UserService } from '../user.service';
```

```
describe('UserService', () => {  
  beforeEach(() => TestBed.configureTestingModule({}));
```

```
  it('should be created', () => {  
    const service: UserService = TestBed.get(UserService);  
    expect(service).toBeTruthy();  
  });
```

```
  it('should return true after late callback', (done: DoneFn) => {  
    const service: UserService = TestBed.get(UserService);  
    service.lateFeedback().then((res: boolean) => {  
      expect(res).toBeTruthy();  
      done();  
    });  
  });  
});
```



Super. Mais on va tout de même attendre 20 s.

# Accélérer le temps

▷ On va appliquer un « wrapper » `fakeAsync` qui nous permettra de manipuler le temps

```
...  
it('should return true after late callback', fakeAsync(() => {  
  const service: UserService = TestBed.get(UserService);  
  
  service.lateFeedback().then((res: boolean) => {  
    expect(res).toBeTruthy();  
  });  
  
  tick(20000); // On joue avec le temps à la prince of Persia  
}));  
...
```

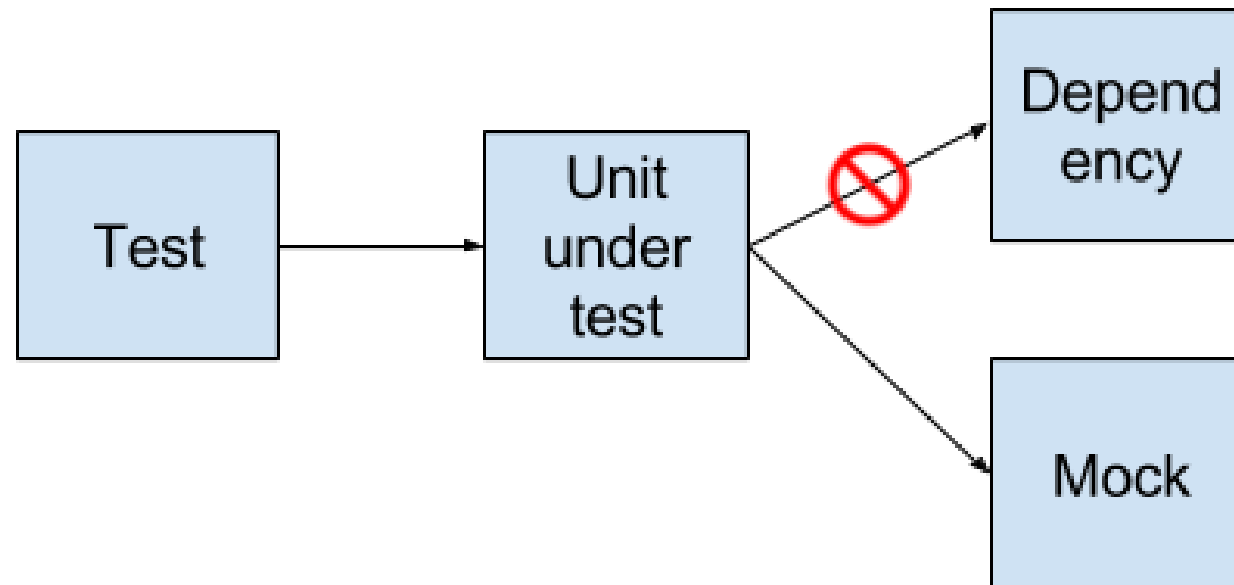


# Mock & Jasmine Spy

- ▷ Un test-unitaire doit être "F.I.R.S.T."
  - › Fast, Independent, Repeatable, Self-Checking, Timely
- ▷ Il faut donc modifier le comportement de certains services (particulièrement les échanges avec les APIs HTTP) à l'aide de Mock et de Spy
- ▷ Nous utiliserons les « `spy` » de jasmine.



# Mock



# Configuration de l'espion

user.spec.service.ts

```
describe('UserService', () => {  
  let service: UserService;  
  let serviceAEspionner: jasmine.SpyObj<ApiHelperService>;  
  
  beforeEach(() => {  
    // Donner le service ainsi que les méthodes à surcharger  
    const spy = jasmine.createSpyObj('ApiHelperService', ['post']);  
  
    // Services à injecter dans l'environnement de test  
    TestBed.configureTestingModule({  
      // On surcharge APIHelper pour appliquer l'espion à la place !  
      providers: [UserService, { provide: ApiHelperService, useValue: spy }]  
    });  
  
    service = TestBed.get(UserService);  
    serviceAEspionner = TestBed.get(ApiHelperService);  
  });  
  
  ...  
});
```

Service qui ira requêter le serveur

Tout va se jouer ici.  
On surcharge le comportement  
du service pour laisser place à l'espion

# Test et configuration du mock

user.spec.service.ts

```
it('should login when good logs passed', (done: DoneFn) => {  
  // On demande à l'espion de renvoyer une certaine donnée SI appel à la méthode "post"  
  serviceAEspionner.post.and.returnValue(  
    new Promise((resolve, rej) => resolve({ success: true })))  
  );  
  
  service  
    .login({ username: 'toto', password: 'peuimporte' })  
    .then((res: boolean) => {  
      expect(res).toBeTruthy();  
      done();  
    });  
});
```

# User Service

user.service.ts

```
@Injectable({
  providedIn: 'root'
})
export class UserService {
  constructor(private api: ApiHelperService) {}

  login({ username, password }: { username: string, password: string }): Promise<boolean> {
    return this.api
      .post('http://34.249.171.19/angular/public/login', { pseudo: username, password })
      .then((res) => res.success);
  }
}
```

# API Helper

api-helper.service.ts

```
@Injectable({
  providedIn: 'root'
})
export class ApiHelperService {
  readonly BASE_URL = 'http://34.249.171.19/angular/public/';


  constructor(private http: HttpClient) {}

  post(action: string, datas: object): Promise<any> {
    return this.http.post(action, datas).toPromise();
  }
}
```

# Karma v4.1.0 - connected

DEBUG

Chrome 77.0.3865 (Windows 10.0.0) is idle

 Jasmine 3.4.0  
• • • • •

[Options](#)

9 specs, 0 failures, randomized with seed 76539

finished in 0.648s

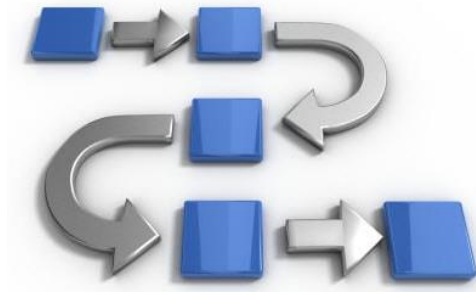
```
ApiHelperService
  • should be created

CalculatorService
  • should say 4 when asked 2 * 2
  • should be created

UserService
  • should be created
  • should login when good logs passed

Calculator
  • should say 4 when asked 2 * 2

AppComponent
  • should have as title 'demo-test'
  • should render title
  • should create the app
```



# Tests End 2 End

# Test End to End

- ▷ Par définition, les tests unitaires ne testent pas les interactions entre les modules et certains effets de bord
- ▷ Protractor est un framework de tests e2e développé par l'équipe d'AngularJS.
  - › Pour **Angular** et AngularJS
- ▷ Avec Protractor il est possible d'implémenter des tests en JavaScript avec une syntaxe **similaire à celle de Jasmine**.
- ▷ Surcouche de **Sélénium**
  - › Utilise WebDriver



# Configuration de protractor

protractor.conf.js

```
exports.config = {
  allScriptsTimeout: 11000, // Définit temps attente protractor le temps qu'Angular soit prêt
  specs: [
    // Chemin vers les fichiers de tests end to end
    './src/**/*.e2e-spec.ts'
  ],
  capabilities: {
    browserName: 'chrome'
  },
  // Si true, Protractor communique directement avec le navigateur sans passer par Selenium (only chrome / firefox)
  directConnect: true,
  // URL sur laquelle protractor va requêter
  baseUrl: 'http://localhost:4200/',
  framework: 'jasmine', // Framework de test utilisé
  jasmineNodeOpts: { // Configuration du framework test
    showColors: true, // Montrer couleur dans terminal
    defaultTimeoutInterval: 30000, // Test échoué si >30s d'attente
    print: function() {} // Format d'affichage des tests
  },
  onPrepare() { // Fonction appelée avant le lancement des tests
    require('ts-node').register({
      project: require('path').join(__dirname, './tsconfig.json')
    });
    jasmine.getEnv().addReporter(new SpecReporter({ spec: { displayStacktrace: true } }));
  }
};
```

# Les Classes .po

items.po

```
export class Items {  
  addElement(elem: string) {  
    element(by.className('itemAdd')).sendKeys(elem);  
    element(by.className('buttonItemAdd')).click();  
  }  
  
  getElements() {  
    return element.all(by.className('item'));  
  }  
}
```

# Test End to End

app.e2e-spec.ts

```
it('should add item', () => {
  page.navigateTo();
  items.addElement('Toto');

  const itemsAdded = items.getElements();
  expect(itemsAdded.count()).toEqual(1);
  expect(itemsAdded.get(0).getText()).toEqual('Toto');
});

afterEach(async () => {
  const logs = await browser.manage().logs().get(logging.Type.BROWSER);
  expect(logs).not.toContain(
    jasmine.objectContaining({
      level: logging.Level.SEVERE
    } as logging.Entry)
  );
});
```

# Questions