formation.tech

Tester des applications Angular

formation.tech

Introduction

Introduction - Formateur

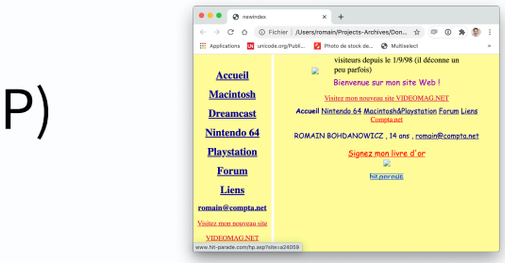
- Romain Bohdanowicz
  Ingénieur EFREI 2008, spécialité en Ingénierie Logicielle

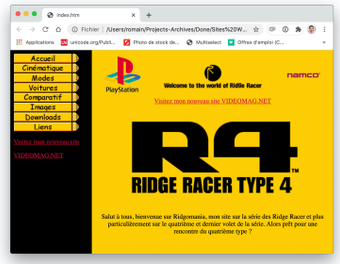
- Expérience
  Formateur/Développeur Freelance depuis 2006
  Près de 2000 jours de formation animées

- Langages
  Expert : HTML / CSS / JavaScript / TypeScript / PHP / Java
  Notions : C / C++ / Objective-C / C# / Python / Bash / Batch

- Certifications
  PHP / Zend Framework / Node.js

- A propos
  Premier site web à 12 ans (HTML/JS/PHP)
  Triathlète du dimanche





Introduction - Horaires

- Matin
  - 9h - 10h
  - 10h15 - 11h15
  - 11h30 - 12h30
- Après-midi
  - 13h45 - 14h45
  - 15h - 16h
  - 16h15 - 17h15

Introduction - formation.tech

- Organisme de formation depuis 2016
- Référencé DataDock
- Certifié Qualiopi
- 15 formations au catalogue
- Une dizaine de formateurs indépendants
- Formations en français ou anglais
- https://formation.tech/

Introduction - WeAreDevs

- Studio de développement créé en 2017
- 1 salarié développeur sénior
- Principales références
  - Cinexpert / Adeum
  - Sponsorise.me
  - Intel
  - Staytuned
  - STMicroelectronics
- https://wearedevs.fr/

Introduction - Et vous ?

- Pré-requis ?
- Rôle dans votre société ?
- Intérêt / objectif de cette formation ?

formation.tech

Tests Automatisés

Tests Automatisés - Introduction

- Comment tester son code ?
  - Manuellement : une personne effectue les tests
  - Automatiquement : les tests ont été programmés
- Historique
  - à partir de 1989 en Smalltalk et le framework SUnit
  - à partir de 1997 en Java avec JUnit
  - à partir de 2004 dans le navigateur avec Selenium

Tests Automatisés - Pourquoi ?

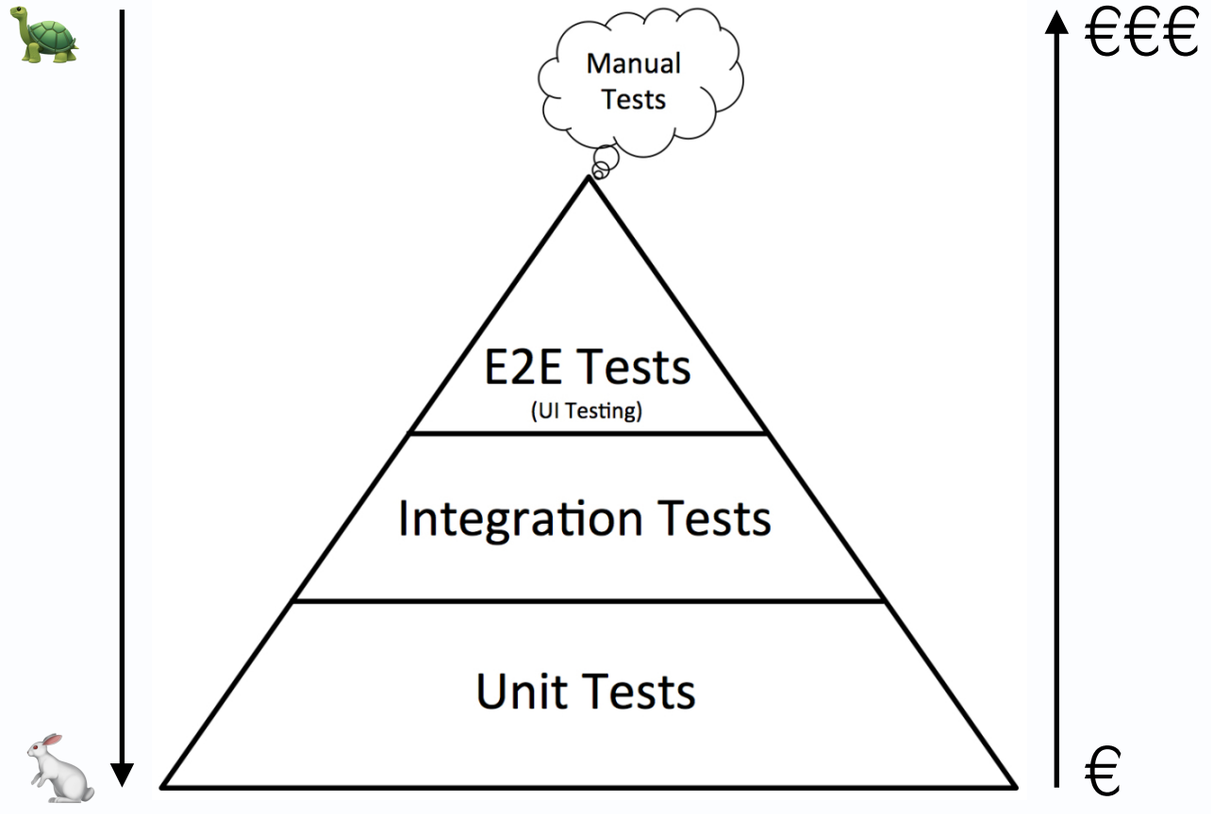
- Pourquoi automatiser les tests ?
  - plus l'application grandit, plus le risque d'introduire une régression est grand
    ex: modifier une fonction qui est partagé par différentes
  - tester manuellement à chaque itération prendra à terme plus de temps qu'écrire le code du test
  - les tests automatisés peuvent se lancer sur différentes plate-formes et navigateurs très simplement
  - les tests aident à la compréhension du code, les lire permet de comprendre des comportements qui n'ont pas toujours été documentés
- Pourquoi tester manuellement ?
  - certains tests peuvent être simple à faire manuellement mais compliqués à automatiser (drag-n-drop...)
  - automatiser permet d'avoir accès à des choses inaccessible manuellement (bouton caché par une popup...)

Tests Automatisés - Types de tests

- Types de tests :
  - tests de code statiques / linters
  - tests de code dynamiques / tests unitaires...
  - tests de déploiement
  - tests de sécurité
  - tests de montée en charge
  - ...

Tests Automatisés - Pyramide des tests

- 3 types de tests automatisés au niveau code côté Front :
  - Test unitaire
    Permets de tester les briques d'une application (classes / fonctions)
  - Test d'intégration
    Teste que les briques fonctionnent correctement ensembles
  - Test End-to-End (E2E)
    Vérifie l'application dans le client
- Un pyramide
  - plus le test est haut plus il est lent
  - plus le test est haut plus il coute cher



Tests Automatisés - Quand exécuter ?

- Quand exécuter ?
  - tout le temps si on arrive à maintenir des tests performants (max 1-2 minutes)
  - avant un commit
  - avant un push
  - sur une plateforme d'intégration ou de déploiement continue (CI/CD)

Tests Automatisés - Organisation

- Ou placer ses tests ?
  - dans le même répertoire que le code testé
  - dans un répertoire *test* en préservant l'arborescence du répertoire *src*
  - dans un répertoire *test* sans lien avec l'arborescence

formation.tech

Jasmine

Jasmine - Introduction

- Créé en 2010
- Licence MIT
- Intègre son propre système de matching et de doubles
- Utilise le style BDD

Jasmine - Mise en place

- Installation
  npm i jasmine -D
- Initialisation
  npx jasmine init
- Lancement des tests
  npx jasmine
- Via le script test
  npm run-script test
  npm run test
  npm test
  npm t

Jasmine - Hello, world

- Jasmine propose 3 fonctions de base
  - describe, permet de définir une suite de tests via un callback
  - it, permet de définir le test
  - expect, permet d'exprimer l'attente (le résultat de ce test devrait être)

```
function sum(a, b) {
  return Number(a) + Number(b);
}


describe('sum function', () => {
  it('should add positive number', () => {
    expect(sum(1, 2)).toEqual(3);
  });
});
```

Jasmine - Hello, world

- Jasmine propose 3 fonctions de base
  - describe, permet de définir une suite de tests via un callback
  - it, permet de définir le test
  - expect, permet d'exprimer l'attente (le résultat de ce test devrait être)

```
function sum(a, b) {
  return Number(a) + Number(b);
}


describe('sum function', () => {
  it('should add positive number', () => {
    expect(sum(1, 2)).toEqual(3);
  });
});
```

Jasmine - Partager des variables entre les tests

- On peut utiliser la portée de closure

```
describe('A suite is just a function', () => {
  let a;

  it('and so is a spec', () => {
    a = true;

    expect(a).toBe(true);
  });
});
```

- Ou bien le mot clé this

```
describe('A suite is just a function', () => {
  this.a;

  it('and so is a spec', () => {
    this.a = true;

    expect(this.a).toBe(true);
  });
});
```

Jasmine - Hooks

- Dans une suite de tests, certaines méthodes seront appelées automatiquement durant la vie du test
  - beforeEach, avant chaque test de la suite (avant chaque *it*)
  - afterEach, après chaque test
  - beforeAll, avant le premier test de la suite (avant le premier *it*)
  - afterAll, après le dernier test de la suite

```
describe('A suite with some shared setup', () => {
  let foo = 0;
  beforeEach(() => {
    foo += 1;
  });
  afterEach(() => {
    foo = 0;
  });
  beforeAll(() => {
    foo = 1;
  });
  afterAll(() => {
    foo = 0;
  });
});
```

Jasmine - Fonction pure

- Les fonctions pures sont les plus simple à tester :
  - elles sont prédictives, appelées avec les mêmes paramètres elles auront toujours le même retour
  - elles sont sans effets de bord (side-effect), elle n'appelle pas d'API externe (réseau, stockage...)
  - elles ne modifient pas leur paramètres d'entrée

```
export function sum(a, b) {
  return Number(a) + Number(b);
}
```

```
import { sum } from './sum';

describe('sum function', () => {
  it('should add positive number', () => {
    expect(sum(1, 2)).toEqual(3);
  });
  it('should convert strings to numbers', () => {
    expect(sum('1', '2')).toEqual(3);
    expect(sum('2', '15')).toEqual(17);
    expect(sum('5', '1')).toEqual(6);
  });
});
```

Jasmine - Tester du code asynchrone

- Avec du code asynchrone le test se termine parfois avant que le code ait été appelé

```
export function asyncCallback(cb) {
  setTimeout(() => {
    cb()
  }, 1000);
}
```

```
import { asyncCallback } from "./async-callback";

describe('asyncCallback function', () => {
  it('should call callback', () => {
    function cb() {
      // le test sera terminé avant l'appel du callback
      expect(true).toBe(false);
    }
    asyncCallback(cb)
  });
});
```

- Pour résoudre le problème on peut :
  - utiliser la fonction done de Jasmine
  - utiliser les promesses
  - utiliser les spies

Jasmine - Tester du code asynchrone

- Avec la fonction done

```
describe('asyncCallback function', () => {
  it('should call callback', (done) => {
    function cb() {
      // ce test échoue
      expect(true).toBe(false);
      done();
    }
    asyncCallback(cb)
  });
});
```

Tests Unitaires sous Angular

Tests Unitaires sous Angular - Introduction

- Par défaut, les tests unitaires et d'intégration sont écrits avec Jasmine et se lancent via Karma
- On les exécute avec la commande :
  `ng test`
- Le module *@angular/core/testing* fournit un certain nombre d'outils de test
- Certains modules liés à des libs également, par exemple
  - *@angular/common/http/testing*
  - *@angular/router/testing*

Tests Automatisés - Test de composant

- Structure d'un test

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';
import { HomeComponent } from './home.component';
import { SharedModule } from '../../shared/shared.module';

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

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ HomeComponent ],
      imports: [ SharedModule ],
    })
    .compileComponents();
  }));

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

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

Tests Automatisés - Test de composant

- Tests d'@Input

```
it('should have correct defaults', () => {
  const fixture = TestBed.createComponent(SelectComponent);
  const select = fixture.debugElement.componentInstance;
  fixture.detectChanges();

  expect(select.opened).toBe(false);
  expect(select.selected).toBe('Rouge');
  expect(select.items).toEqual(['Rouge', 'Vert', 'Bleu']);
});

it('should contains selected @Input', () => {
  const fixture = TestBed.createComponent(SelectComponent);
  const select: SelectComponent = fixture.debugElement.componentInstance;

  select.selected = 'Toto';

  fixture.detectChanges();

  const compiled = fixture.debugElement.nativeElement;
  expect(compiled.querySelector('.selected').textContent).toContain('Toto');
});
```

Tests Automatisés - Test de composant

- Tests d'@Ouput

```
it('should emit selectedChange @Ouput', () => {
  const fixture = TestBed.createComponent(SelectComponent);
  const select: SelectComponent = fixture.debugElement.componentInstance;

  select.opened = true;

  fixture.detectChanges();

  select.selectedChange.subscribe((selected) => {
    expect(selected).toBe('Bleu');
  });

  const compiled: HTMLElement = fixture.debugElement.nativeElement;

  const event = document.createEvent('MouseEvent');
  event.initEvent('click');
  compiled.querySelector('.item:last-child').dispatchEvent(event);
});
```

Tests Automatisés - Test de composant

- Tests avec Mock du Backend HTTP
  Il faudra alors importer *HttpClientTestingModule*

```
it('should contains users', () => {
  httpTestingController.expectOne('/users').flush([{id: 123, name: 'Titi'}]);

  fixture.detectChanges();
  const compiled: HTMLElement = fixture.nativeElement;

  expect(compiled.querySelector('.list-group-
item').textContent).toContain('Titi');

  httpTestingController.verify();
});
```

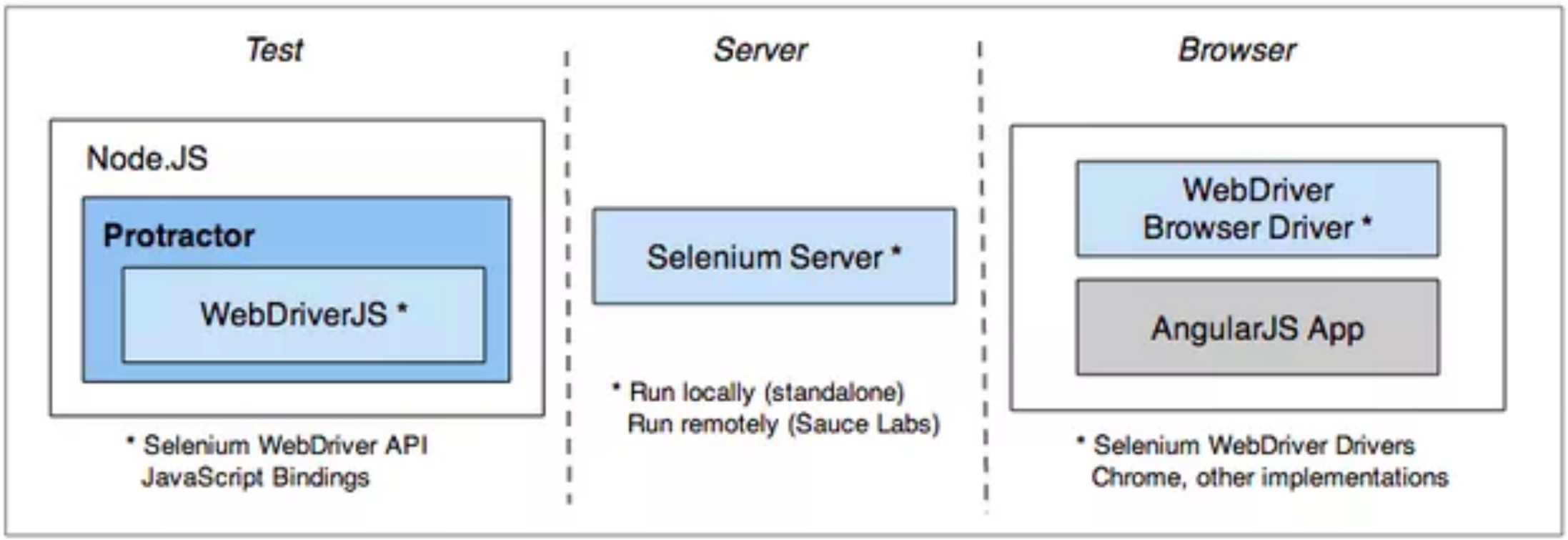
Tests E2E

Tests E2E - Introduction

- Les tests E2E utilisent Protractor (surchouche de Selenium)
- On les exécute avec la commande :
  `ng e2e`
- Depuis Angular 12, Protractor n'est plus installé par défaut pour laisser le choix d'utiliser d'autres bibliothèques de test, par exemple
  - Cypress
  - Webdriver.io
  - CodeceptJS
  - CucumberJS
  - Puppeteer
  - ...

Tests E2E - Prérequis

- Protractor nécessite l'utilisation de WebDriverJS, c'est à dire la version de Selenium normée par le W3C
- Il faudra installer des drivers pour exécuter dans les navigateurs



Tests E2E - Page Object

- Lors de l'écriture de Test E2E on privilégiera l'utilisation du Pattern Page Object
- Pour cela nous allons créer une classe contenant des méthodes pour intéragir avec une certaine page, par exemple pour un formulaire de login
  - navigateToLoginPage
  - fillUsername
  - fillPassword
  - clickOnSubmitButton
  - getErrorsDiv
  - ...

Tests E2E - Page Object

- Exemple de Page Object

```
import { browser, by, element } from 'protractor';

export class AppPage {
  navigateTo(): Promise<unknown> {
    return browser.get(browser.baseUrl) as Promise<unknown>;
  }

  getTitleText(): Promise<string> {
    return element(by.css('app-root .content span')).getText() as Promise<string>;
  }
}
```

Tests E2E - Exemple de test

- Protractor utilise Jasmine comme framework de test
- On peut ainsi utiliser les fonctions describe, it, beforeEach, afterEach, expect...

```
import { browser, logging } from 'protractor';

import { AppPage } from './app.po';

describe('workspace-project App', () => {
  let page: AppPage;

  beforeEach(() => {
    page = new AppPage();
  });

  afterEach(async () => {
    // Assert that there are no errors emitted from the browser
    const logs = await browser.manage().logs().get(logging.Type.BROWSER);
    expect(logs).not.toContain(jasmine.objectContaining({
      level: logging.Level.SEVERE,
    } as logging.Entry));
  });
});
```