Facebook Jest

Jest ile JavaScript Testleri

Hazırlayan:
Ali GÖREN
https://aligoren.com

Neleri İnceleyeceğiz?

- Birim Test Nedir?
- Birim Test Ne Zaman Yapılmalıdır?
- Doğru Birim Test Nasıl Yazılır?
- JavaScript Birim Test Kütüphaneleri
- Jest vs Mocha
- Jest Nasıl Kullanılır
- Kaynaklar

Birim Test Nedir?

Birim Test Nedir?

- Birim Test, yazılım birimlerinin test edilmesidir.
- Yazılım birimi, yazılımda bulunan, test edilebilecek en küçük bileşendir.
- Birim testler, yazılımların ürün olmadan önceki son halinin, beklentilere uygun çalışıp çalışmadığının test edilmesi amacıyla geliştirilir.
- Birim testler, geliştirme işlemini yapan developer tarafından yapılır.
- Birim testler, hataları bulmak için değildir

Birim Test Ne Zaman Yapılmalıdır?

Birim Test Ne Zaman Yazılmalıdır

- Birim testler, yazılım geliştirme süreçlerinin sonunda yazılabilirler
- Birim testler, her sprintin sonunda yazılabilir
- Henüz ortada kod olmayan durumlarda, TDD yöntemi seçilerek birim testler tercih edilebilir.
- TDD yönteminde, kodlar test senaryolarına göre yazılır, böylelikle refactoring işlemi daha kolay hale gelir.

Doğru Birim Test Nasıl Yazılır?

Doğru Birim Test Nasıl Yazılır?

- Her test, tek bir şeyi test etmelidir. Gerekmedikçe aynı test birden fazla assert ifadesi içermemelidir.
- Sınıfları bağımlılıklarından izole edin.
- Yazılan testler birbirleri ile bağlantılı olmamalı.
- Yazacağınız testler doğru olarak isimlendirilmeli.
- Test kodlarını okunabilir yazın. Tekrarlanmış test kodlarından kaçının.

JavaScript Birim Test Kütüphaneleri

Jasmine

Jasmine, Angular ile entegre olan, kurulumu ve kullanımı kolay JavaScript Testing Frameworktür.

Web Sitesi: https://jasmine.github.io/

AVA

AVA, minimilastic testing frameworktür. Avantajı, JavaScript’in async özelliğini kullanmasıdır. Kurulumu ve kullanımı kolaydır

Web Sitesi: https://github.com/avajs/ava

Mocha

Mocha, çok kullanılan ve community’si bir hayli yüksek olan, kullanım kolaylığına sahip testing frameworktür.

Web Sitesi: https://mochajs.org/

Mocha

- Mocha, Jest’e göre daha eski olduğu için, Jest’in henüz yaşamakta olduğu ve yaşayacağı bazı problemleri çoktan atlatmış olabilir.
- Mocha, esnektir. Herhangi bir assertion kütüphanesi ya da mocking kütüphanesiyle direkt gelmez. Neye ihtiyaç duyarsanız onu seçersiniz.

Jest

Jest, Facebook tarafından geliştirilen, testing frameworktür.
Communitysinin geniş olmasının yanı sıra Facebook ve büyük şirketlerce kullanılması bir avantaj sayılabilir.

Web Sitesi: https://facebook.github.io/jest/

Jest vs Mocha

Jest

- Jest’in en büyük avantajı, minimal basit kurulum ve ayarlamalara sahip olmasıdır.
- Testler, diğer Birim Test Kütüphanelerinde olduğu gibi BDD yöntemiyle yazıldı.
- Testleri __tests__ klasörü altında ya da *.spec.js ya da *.test.js dosya isimleriyle oluşturabilirsiniz.
- Jest Snapshot Desteğine Sahiptir

Jest

- Jest Mocha ve benzeri kütüphanelere göre henüz yeni sayılabileceğinden, toolları ve destekleyen editör, IDE gibi araçları da daha azdır.
- Jest bazı IDE’ler tarafından henüz debuggerlarda desteklenmeyebiliyor.
- Entegrasyon testleri Mocha’ya göre daha zayıftır.

Mocha

- Mocha, Jest’e göre daha eski olduğu için, Jest’in henüz yaşamakta olduğu ve yaşayacağı bazı problemleri çoktan atlatmış olabilir.
- Mocha, esnektir. Herhangi bir assertion kütüphanesi ya da mocking kütüphanesiyle direkt gelmez. Neye ihtiyaç duyarsanız onu seçersiniz.
- Editör ve IDE esteği mevcuttur
- Mocha’nın community’si bir hayli büyüktür.

Mocha

- Mocha’nın asıl sorunu, kolay kullanımına karşın çok fazla ayarlama gerektirmesidir.
- Esneklik, kimi zaman Mocha için sorun oluşturabilir çünkü assertion, mocking ve coverage işlemleri için tek tek ayarlamalar yapmak zorunda kalabilirsiniz.
- Esneklik iyi değerlendirilebilirse mükemmel olur, ancak hatalı değerlendirilirse kabusa dönüşebilir.
- Mocha’da snapshotlar kullanılabilir ancak entegrasyonu faciaya dönüşebilir.

Jest Nasıl Kullanılır

Jest Nasıl Kullanılır

Jest NPM ya da YARN kullanılarak kurulabilir:

NPM

```
npm install --save-dev jest
```

YARN

```
yarn add --dev jest
```

Jest Nasıl Kullanılır

İlk basit testimiz için öncelikle **sum.js** adında bir dosy oluşturalım. Bu dosya, iki sayıyı toplayan fonksiyonu barındıracaktır.

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

module.exports = sum;
```

Yukarıda bulunan toplama fonksiyonunu modül olarak dışarıya export ettik. Fark ettiyseniz herhangi bir şekilde başka bir kütüphane ile bağlı değil.

Jest Nasıl Kullanılır

Yazdığımız modül için **sum.test.js** adında bir dosya oluşturalım ve içine test kodlarımızı yazalım:

```
const sum = require('./sum');

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

Yazılan kod, bize 1 ve 2 toplamının 3’e eşit olduğunu bekleyen bir test olduğunu anlatıyor.

Bu kısımda test fonksiyonu, teste dair açıklamayı barındırırken, expect ise value yani değeri barındırır. toBe fonksiyonu ise değerin ne olması gerektiğini beklediğini bildirir.

Jest Nasıl Kullanılır

Testlerimizi yazdıktan sonra, **package.json** dosyası altına test için hangi aracı kullanacağımızı bildirmeliyiz.

```
{
  "scripts": {
    "test": "jest"
  }
}
```

Bu aşamadan sonra aşağıdaki komutlardan herhangi birisi testleri çalıştırmanıza yardımcı olacaktır:

- npm t
- npm test
- npm run test

```
PASS  ./sum.test.js
✓ adds 1 + 2 to equal 3 (5ms)
```

Yukarıdaki sonuç testin, geçip geçmediğini bize bildiren bir sonuç döndürecektir.

Jest Nasıl Kullanılır

Jest birden fazla matchers'a sahiptir. Ancak, diğer frameworklerin de üzerinde durduğu en çok kullanılanları önermektedir. En çok kullanılanlardan ilki işlem ve beklenene dair bir örnekle açıklanabilir:

```
test('two plus two is four', () => {
  expect(2 + 2).toBe(4);
});
```

toBe tam eşitliği sağlamak için `===` kullanır. Eğer bir nesnenin değerini kontrol etmek isterseniz **toEqual** kullanmalısınız.

```
test('object assignment', () => {
  const data = {one: 1};
  data['two'] = 2;
  expect(data).toEqual({one: 1, two: 2});
});
```

Jest Nasıl Kullanılır

Eğer, matchers(eşleştirici) tam tersini kontrol etmek isterseniz **not** kullanabilirsiniz:

```
test('adding positive numbers is not zero', () => {
  for (let a = 1; a < 10; a++) {
    for (let b = 1; b < 10; b++) {
      expect(a + b).not.toBe(0);
    }
  }
});
```

Bütün eşleştiricilerde not kullanabilirsiniz ancak, anlam karmaşası yaşanmaması adına uygun noktalarda kullanmanız daha faydalı olacaktır.

Jest Nasıl Kullanılır

Truthiness Kavramı

Jest ile bazı durumlarda ayırıcı özellikleri kullanarak testler gerçekleştirebilirsiniz. Bunlar undefined, null ya da false olabilir. Jest spesifik türler üzerinde ya da değerler üzerinde testler gerçekleştirmenize imkan tanıyan bazı helperlara sahiptir. Bunlar şöyledir:

- toBeNull => Sadece null değerlerle eşleşecektir.
- toBeUndefined => Sadece undefined türünden değerlerle eşleşecektir.
- toBeDefined => toBeUndefined'ın tam karşıtı olan değerlerle eşleşecektir.
- toBeTruthy => if ifadesinde ne olursa olsun true dönen değerlerle eşleşecektir.
- toBeFalsy => if ifadesinde ne olursa olsun false dönen değerlerle eşleşecektir.

```
test('null', () => {
  const n = null;
  expect(n).toBeNull();
  expect(n).toBeDefined();
  expect(n).not.toBeUndefined();
  expect(n).not.toBeTruthy();
  expect(n).toBeFalsy();
});

test('zero', () => {
  const z = 0;
  expect(z).not.toBeNull();
  expect(z).toBeDefined();
  expect(z).not.toBeUndefined();
  expect(z).not.toBeTruthy();
  expect(z).toBeFalsy();
});
```

Jest Nasıl Kullanılır

Sayılar

Jest ile test gerçekleştirirken, sayıların kontrolünü kolaylıkla sağlayabilirsiniz. Büyüklük, küçüklük, eşitlik ya da büyük ve küçük eşit durumlarını kontrol edebilirsiniz:

```
test('two plus two', () => {
  const value = 2 + 2;
  expect(value).toBeGreaterThan(3);
  expect(value).toBeGreaterThanOrEqual(3.5);
  expect(value).toBeLessThan(5);
  expect(value).toBeLessThanOrEqual(4.5);
  // toBe and toEqual are equivalent for numbers
  expect(value).toBe(4);
  expect(value).toEqual(4);
});
```

Bazı durumlarda yuvarlama ile ilgili sorunlar yaşama riskiniz var. Eğer floating point eşitliğini kontrol ediyorsanız toBeCloseTo fonksiyonu sizin için idealdir. Sizi ufak tefek yuvarlama işlerinden kurtarır.

```
test('adding floating point numbers', () => {
  const value = 0.1 + 0.2;
  expect(value).not.toBe(0.3);    // It isn't! Because rounding error
  expect(value).toBeCloseTo(0.3); // This works.
});
```

Jest Nasıl Kullanılır

Stringler

String ifadeleri toMatch fonksiyonu ile test edebiliriz. Bu işlem aşamasında RegExp kullanabiliriz:

```
test('there is no I in team', () => {
  expect('team').not.toMatch(/I/);
});

test('but there is a "stop" in Christoph', () => {
  expect('Christoph').toMatch(/stop/);
});
```

Yukarıda yer alan kodda gördüğünüz gibi not ifadesini toMatch ile kullanabilmekteyiz. Ayrıca, toMatch fonksiyonunu RegExp ile kullanabiliyoruz.

Jest Nasıl Kullanılır

Arrayler

Jest, arrayler içerisinde yer alan bazı stringleri ya da sayıları kontrol etmemize imkan tanır. Bu işlemi yaparken toContain fonksiyonunu kullanırız. Böylelikle testlerde arrayleri de kontrol edebiliriz:

```
const shoppingList = [
  'diapers',
  'kleenex',
  'trash bags',
  'paper towels',
  'beer',
];

test('the shopping list has beer on it', () => {
  expect(shoppingList).toContain('beer');
});
```

Array içerisinde bir test gerçekleştirmek istiyorsak bunu toContain fonksiyonunu kullanarak yapabileceğimizi gördük.

Jest Nasıl Kullanılır

Exceptionlar

Jest, exceptionları yani oluşabilecek hataları yakalama yeteneğine sahiptir. Bu konuya dair testler yaparken toThrow fonksiyonunu kullanırız.

```
function compileAndroidCode() {
  throw new ConfigError('you are using the wrong JDK');
}

test('compiling android goes as expected', () => {
  expect(compileAndroidCode).toThrow();
  expect(compileAndroidCode).toThrow(ConfigError);
  // You can also use the exact error message or a regexp
  expect(compileAndroidCode).toThrow('you are using the wrong JDK');
  expect(compileAndroidCode).toThrow(/JDK/);
});
```

Ayrıca toThrow fonksiyonu RegExp ile çalışabilir. Böyleleri hataları ayıklayarak testler gerçekleştirebilirsiniz.

Tüm matcherların (eşleştiriciler) listesini https://facebook.github.io/jest/docs/expect.html adresinden kontrol edebilirsiniz.

Kaynaklar

Kaynaklar

- https://raygun.com/blog/javascript-unit-testing-frameworks/
- https://spin.atomicobject.com/2017/05/02/react-testing-jest-vs-mocha/
- http://www.canertosuner.com/post/unit-test-nedir
- http://www.gokhan-gokalp.com/unit-test-yazarken-kolay-mock-islemleri/
- http://www.burakavci.com.tr/2017/01/unit-testing.html
- http://www.onurarslan.org/yazilim-birim-testiunit-test-nedir/
- https://www.seckintozlu.com/1068-birim-test-nedir-nicin-yapilir-nasil-yapilir.html
- https://facebook.github.io/jest/

HAZIRLAYAN

Ali GÖREN

goren.ali@yandex.com

https://aligoren.com

https://github.com/aligoren