React ile Test Yazımı

Testing

Konu Başlıkları

1.Test Türleri

2.Jest

3.React Testing Library

4.Test Dosya Yapısı

5.Snapshot

6.Mock (msw)

Test

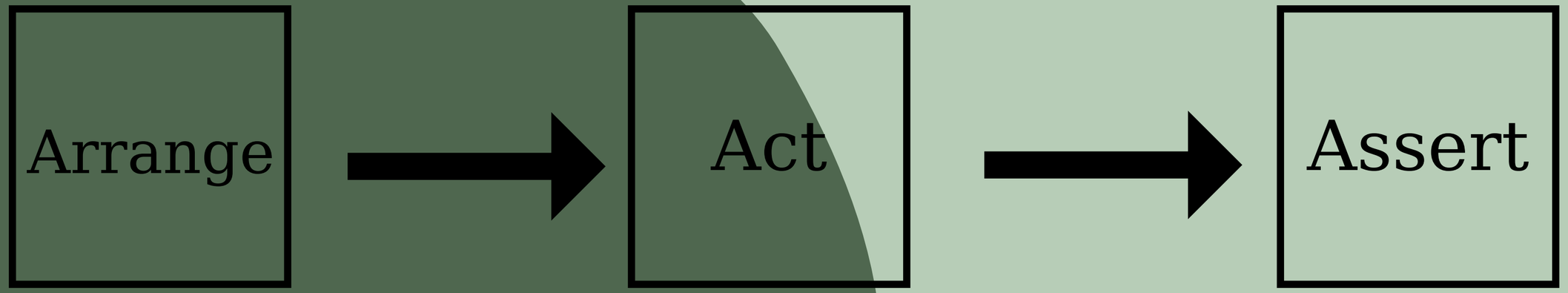
Test genelde üç aşamadan oluşan, bir senaryo belirlediğimiz bu senaryodaki duruma bir girdi verdiğimiz ve sonucunda istediğimiz çıktıyı vermesini beklediğimiz işlemler bütünüdür. Test yazarken ana amacımız verdiğimiz senaryonun başarıyla sonuçlanmasıdır.

Test türlerini üç ana başlıkta toplayabiliriz:

1- Unit test: Uygulamamızda test edebileceğimiz en küçük birim diyebiliriz. Tek bir fonksiyon modül, object vs test edilebilir. Kodumuzun bir bölümünü izole hale getirip istediğimiz gibi çalışıp çalışmadığını kontrol edebiliriz.

2- Integration test: Farklı parçaların birbiriyle uyumlu çalışıp çalışmadığını test edebilmek için uyguladığımız test türüdür.

3- End to End test: Uçtan uca test olarak bilinir. Unit ve integration testlerimizin kombinasyonuyla uygulamamızın uçtan uca doğru çalışıp çalışmadığını test etmek için uygulanır.



Arrange: uygulamanın belli bir durumda olması yani sonraki adımlara hazır olması için ilk etapta kodlarınızı ayarlarsınız

Act: Hareket geçme (click,input).Eyleme geçersiniz (act), bir kullanıcının yapması gereken adımları gerçekleştirirsiniz (tıklama gibi)

Assert: Uygulamanın yeni durumu iddia edilir. Mevcut durumlar test edildiğinde hipotez doğruysa test geçilir değilse fail olur.

Jest

Jest, başlangıçta facebook tarafından özellikle React uygulamalarını test etmek için oluşturuldu. Ancak tanıtıldığından beri oldukça popülarite kazanmış ve sadece React ile değil diğer bir çok library ve framework ile de çalışabilir hale gelmiştir. Jest'in bazı faydalı özelliklerini sayacak olursak: Dökümantasyonu ve syntax'ı çok kolaydır. Ayrıca hiçbir config ayarı yapmadan kolaylıkla test yazmamıza olanak sağlar.

```
yarn add --dev jest
# or
npm install --save-dev jest
```

React Testing Library

React Testing Library Kent C. Dodds tarafından enzyme alternatifi olarak oluşturulmuş ve react komponentlerini test edebilmek için kullanabileceğimiz bir test librarysidir. Herhangi bir framework ile uyumlu çalışır.

Eğer Create React App kullanıyorsanız herhangi bir şey yapmanıza gerek yok çünkü içerisinde halihazırda eklenmiş olarak geliyor ancak kendiniz eklemek isterseniz de npm kullanarak aşağıdaki komutu terminale yazarak projenize ekleyebilirsiniz:

npm i --save-dev @testing-library/react

The more your tests resemble the way your software is used,
the more confidence they can give you.

Sıfırdan Proje İnceleme

Matchers

Matchers (eşleştiriciler) testlerinizin sonuçlarını ve değerlerini farklı şekillerde eşleştirerek doğrulamanıza izin verir. Örneğin bir faktöriyel hesaplama fonksiyonumuzun olduğunu varsayalım. expect() metodunu ve basit bir matcher kullanarak işlemimizin sonucunun beklediğimiz değer olup olmadığını kontrol edebiliriz.

toBe: primitive değerleri (boolean, number, string) veya objelerin ve arraylerin referanslarını (diğer adıyla referans eşitliği) karşılaştırır.

toEqual: Array'lerin veya objelerin tüm özelliklerini (diğer bir deyişle deep equality) karşılaştırır.

toBeTruthy, (toBeFalsy): değerlerin true yada false olup olmadığını belirtir.

not: bir matcher önüne yerleştirilmelidir ve matcher'ın sonucunun tersini döndürür.

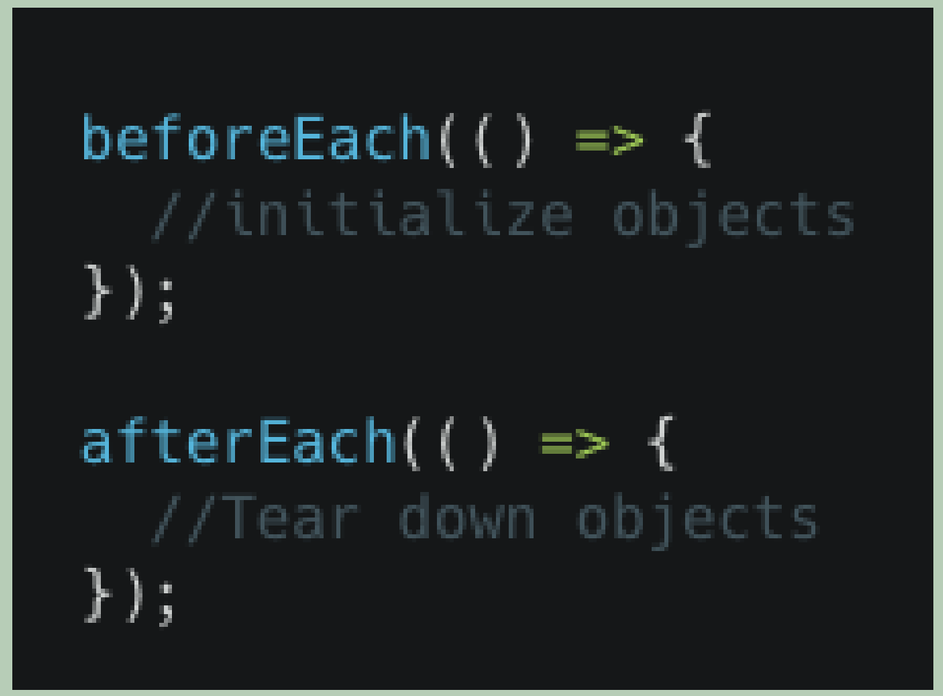
https://jestjs.io/docs/expect#methods

Mocking

Uygulama içerisinde veritabanı, network erişimi, dosya erişimi vb. dışa bağımlı kısımlar bulunabilir. Bu kısımlar test edildiğinde, testlerin çalışma sürelerini uzatacakları için, bu kısımların fake (mock) versiyonları oluşturulur.Bir nesnenin davranışını izole etmek için, etkileşime girdiği diğer nesneleri gerçek nesnelerin davranışını simüle eden mock (taklit) nesnelerle değiştirmek gerekebilir. Kısacası Mocking gerçek dataların davranışını simüle etmek için kullandığımız fake durumlardır. Bu sayede yüzlerce testiniz olsa dahi çok hızlı bir biçimde çalıştırabilirsiniz.

Lifecycle

Bazen test işlemleri sırasında birbiri ardına testler çalıştırma ve o testlerden önce veya sonra her defasında yapılması gereken işlere ihtiyaç duyarız. Çoğu framework'ler bu gibi konular için lifecycle fonksiyonları kullanırlar.



Test Dosyasını Yapısı

Jest, testlerinizi yapılandırmak için bazı fonksiyonlar sağlar

describe: testlerinizi gruplamak ve fonksiyon, modül ya da class'ın davranışını açıklamak için kullanılır. İki parametre alır. İlki, grubunuzu tanımlayan bir string değer, ikincisi, test senaryolarınıza sahip olduğunuz bir callback fonksiyondur.

it ya da test: bu sizin test durumunuz, yani unit testinizdir. Mümkün olduğu kadar açıklayıcı olmalı. Parametreler describe ile aynıdır.

```
describe("My first test suite", () => {
  it("adds two numbers", () => {
    expect(add(2, 2)).toBe(4);
  });

  it("substracts two numbers", () => {
    expect(substract(2, 2)).toBe(0);
  });
});
```

```
test("username input should be rendered",()=>{
    const usernameInputEl=screen.getByPlaceholderText
    expect(usernameInputEl).toBeInTheDocument();
});
```

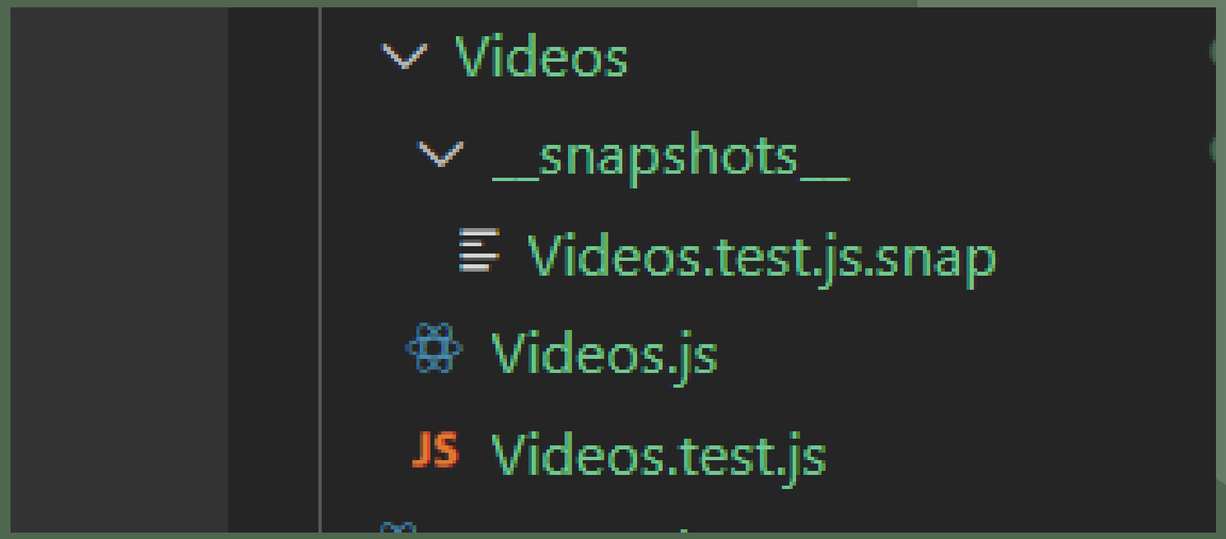
Counter,Buton,Login Component

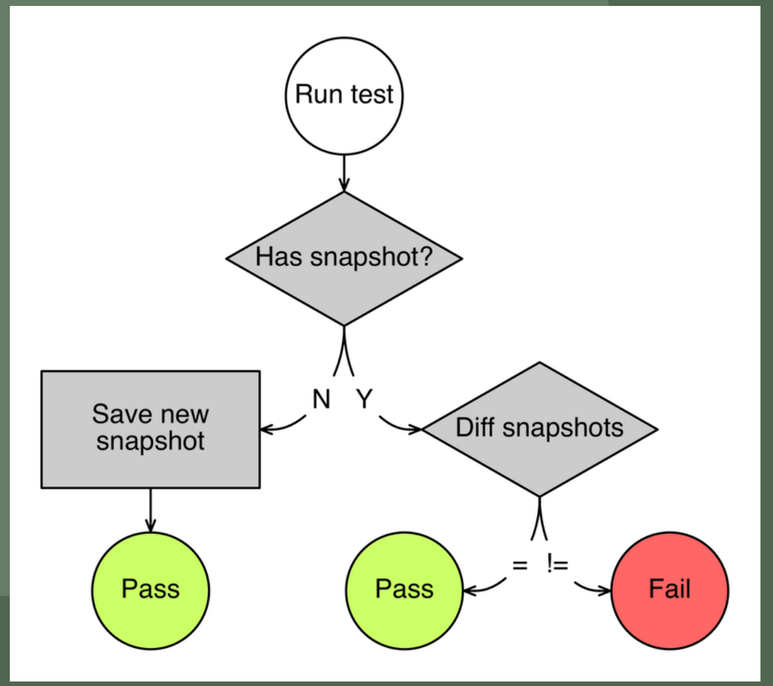
SnapShot

Snapshot testleri aslında ekranın bir görüntüsünü alıp kaydederek daha sonra kıyaslamak için kullanılmasıdır. Çünkü snapshot testlerinde UI bileşeni render edilir ve bir dosya içerisine kaydedilir. Test sonraki çalıştırmada kaydedilmiş dosyanın içeriğini okur ve yeni render ettiği UI bileşeni ile karşılaştırır.

Sizin için otomatik olarak bir __snapshots__ klasörü ve test.js.snap dosyası oluşturulacaktır. Sonraki her testte yeni anlık görüntü, mevcut anlık görüntü dosyasıyla karşılaştırılıyor. Anlık görüntü değişmediyse test başarılı olacak ve değiştiyse başarısız olacaktır.

npm i react-test-renderer






```
Test Suites: 5 passed, 5 total
Tests:       33 passed, 33 total
Snapshots:   3 passed, 3 total
Time:        4.63 s
Ran all test suites.

Watch Usage: Press w to show more.
```

Videos Component

Mock

Mock ile gerçek bir objenin davranışını simüle eden sahte bir modül yazabiliriz. Başka bir deyişle, mock, test ettiğimiz şeyi izole etmek için kodumuzu taklit etmemize izin verir. Aşağıdaki fonksiyonu kullanarak kolayca mock fonksiyonlar oluşturabilirsiniz:

jest.fn()

Elbette, mock kullanımı sadece fonksiyonlarla sınırlı değildir. Modüllerle birlikte de mock yapısını kullanabiliriz.

```
jest.mock("axios",()=>({
    __esModule:true,

    default:{
        get: ()=>({
            data:{id:1,name:"John"}
        })
    }
```

https://jestjs.io/docs/expect#tohavebeencalled

LoginValidation,LoginMock Component

Mock Service Worker

Service workers tarayıcınızın arka tarafında çalışan sayfadan bağımsız scriptlerdir. Service workers sayfanıza hiç bir şekilde müdahele edemez sadece sayfa içerisinde bulunan script ler ile iletişime geçerek dolaylı yoldan bir müdahalede bulunabilir.Network üzerinden gelen istekleri engelleyebilir cevap verebilir yada farklı bir yere yönlendirebilme olanağı sunar ve offline durumda da çalışabilir.

Burada fetch event ile network üzerinde giden istekleri yakalayıp yönlendirebilir, engelleyebilir yada farklı bir şekilde cevap verebiliriz. Örnektede anlatacağım üzere request i url ile kontrol sağlayıp kendi response object imiz ile cevap veriyoruz.

npm i msw --save-dev

TodoList Component