Component Testing

Vitest ve React Testing Library kurulumu, render+screen ile ilk test, getByRole /getByLabelText ile sorgulama, jest-dom matcher'ları, ve koşullu render'ı test etmek -- basit örneklerle.

Orta 12 dk

Son güncelleme: 6 Ekim 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/component-testing

---

Component Testing

Şimdiye kadar yazdığımız her component'i TARAYICIDA elle tıklayarak kontrol ettik. Bu, birkaç component için yeterli olsa da, uygulama büyüdükçe her değişiklikten sonra her ekranı elle kontrol etmek hem yavaş hem de güvenilmez. Bu ders, component'lerin doğru çalıştığını OTOMATİK olarak, kod ile doğrulamayı öğretiyor.

Vitest ve React Testing Library Kurulumu

Bu kursta iki kütüphane kullanıyoruz:

- **Vitest** — testleri ÇALIŞTIRAN araç (`describe` , `it` , `expect` gibi fonksiyonları sağlar). Vite tabanlı projeler için tasarlandığı için ek bir yapılandırmaya neredeyse hiç ihtiyaç duymaz.
- **React Testing Library (RTL)** — component'leri sahte bir DOM'a (jsdom) YERLEŞTİRİP, o DOM'u gerçek bir kullanıcının göreceği şekilde SORGULAMAMIZI sağlayan kütüphane.

Bir Vite projesine eklemek için:

```
npm install -D vitest @testing-library/react @testing-library/jest-dom jsdom
```

`vite.config.js` içine bir `test` bloğu eklenir:

```
export default defineConfig({
  plugins: [react()],
  test: {
    environment: "jsdom",
    setupFiles: ["./src/setupTests.js"],
    globals: true,
  },
});
```

`environment: "jsdom"` testlerin gerçek bir tarayıcı yerine, Node içinde ÇALIŞAN sahte bir DOM'da koşmasını sağlar. `setupFiles` içindeki dosyada tek bir satır yeterli:

```
import "@testing-library/jest-dom/vitest";
```

Bu satır, birazdan göreceğimiz `toBeInTheDocument()` gibi ek doğrulamaları (matcher) Vitest'in `expect`'ine EKLER.

render() ve screen ile İlk Testimiz

Bir testin en temel iskeleti; component'i sahte DOM'a yerleştirmek ve içinde beklediğimiz bir şeyin olduğunu doğrulamaktan oluşur:

```
import { render, screen } from "@testing-library/react";
import { describe, it, expect } from "vitest";

// The component under test. In a real project this usually lives in its own
// file (Counter.jsx), with the test written in a separate file
// (Counter.test.jsx) -- here we've combined the two into a single readable example.
function Counter() {
  return (
    <div>
      <p>Count: 0</p>
    </div>
  );
}

describe("Counter", () => {
  it("renders the initial count", () => {
    // render() places the component into a real DOM (jsdom, a SIMULATED browser).
    render(<Counter />);

    // screen is used to QUERY the current DOM. If getByText can't find an
    // element containing exactly this text, it fails the test IMMEDIATELY.
    expect(screen.getByText("Count: 0")).toBeInTheDocument();
  });
});
```

`describe`, ilgili testleri bir grup altında toplar; `it` (ya da `test`), tek bir test senaryosunu tanımlar. `render(<Counter />)`, component'i jsdom'a yerleştirir. `screen`, o anki DOM'u SORGULAMAK için kullanılır -- `getByText`, verilen metni içeren bir eleman bulamazsa test ANINDA başarısız olur.

getByRole ve getByLabelText ile Sorgulama

`getByText` her zaman en doğru sorgu değildir -- RTL, gerçek kullanıcıların (ve ekran okuyucuların) sayfayı nasıl ALGILADIĞINA daha yakın sorgular sunar:

```
import { render, screen } from "@testing-library/react";
import { describe, it, expect } from "vitest";

function LoginButton() {
  return <button>Log In</button>;
}

function NameField() {
  return (
    <div>
      <label htmlFor="name">Name</label>
      <input id="name" defaultValue="Ada" />
    </div>
  );
}

describe("Querying elements", () => {
  it("finds a button by its accessible role and name", () => {
    render(<LoginButton />);

    // getByRole finds elements by their ACCESSIBILITY role, not by their
    // VISIBLE text -- a <button> has the "button" role. This is the query
    // style that most closely matches how real users (and screen readers)
    // perceive the page.
    expect(screen.getByRole("button", { name: /log in/i })).toBeInTheDocument();
  });

  it("finds a form field by its connected label", () => {
    render(<NameField />);

    // getByLabelText finds the input that matches a <label htmlFor="...">
    // -- there's no need to add the input's id or a test-id.
    expect(screen.getByLabelText("Name")).toHaveValue("Ada");
  });
});
```

`getByRole("button", { name: /log in/i })`, bir `<button>` elemanını ERİŞİLEBİLİRLİK rolünden ve görünen adından bulur -- RTL'in resmî dokümantasyonu, mümkün olduğunda `getByRole`'ü ÖNCELİKLİ sorgu olarak önerir. `getByLabelText("Name")`, `<label htmlFor="name">` ile eşleşen input'u, id veya test-id eklemeye gerek kalmadan bulur.

jest-dom Matcher'ları

Kurulumda eklediğimiz `@testing-library/jest-dom/vitest`, `expect`'e DOM'a özel yeni doğrulamalar ekler:

```
import { render, screen } from "@testing-library/react";
import { describe, it, expect } from "vitest";

function SubmitButton({ disabled }) {
  return <button disabled={disabled}>Submit</button>;
}

describe("SubmitButton", () => {
  it("is disabled when the disabled prop is true", () => {
    render(<SubmitButton disabled={true} />);

    // toBeDisabled/toBeEnabled/toBeInTheDocument are matchers added by
    // @testing-library/jest-dom -- they're not in plain Vitest, and are
    // installed separately in projects that use jsdom (via
    // "@testing-library/jest-dom/vitest" in setupFiles).
    expect(screen.getByRole("button", { name: /submit/i })).toBeDisabled();
  });

  it("is enabled when the disabled prop is false", () => {
    render(<SubmitButton disabled={false} />);

    expect(screen.getByRole("button", { name: /submit/i })).toBeEnabled();
  });
});
```

`toBeDisabled()` ve `toBeEnabled()`, bir elemanın `disabled` özniteliğini kontrol eder; `toBeInTheDocument()` bir elemanın DOM'da var olup olmadığını doğrular. Bunlar düz Vitest'te YOKTUR -- jest-dom paketinin eklediği, DOM testleri için özel olarak tasarlanmış matcher'lardır.

Koşullu Render'ı Test Etmek

State ve Event'ler kategorisinde gördüğümüz koşullu render deseni, en sık test edilen senaryolardan biridir -- her durumun DOĞRU metni gösterdiğini ayrı ayrı doğrularız:

```
import { render, screen } from "@testing-library/react";
import { describe, it, expect } from "vitest";

// A tested version of the conditional rendering pattern from the State & Events lesson.
function StatusMessage({ status }) {
  if (status === "loading") return <p>Loading...</p>;
  if (status === "error") return <p>Something went wrong.</p>;
  return <p>Data loaded successfully.</p>;
}

describe("StatusMessage", () => {
  it("shows a loading message", () => {
    render(<StatusMessage status="loading" />);
    expect(screen.getByText("Loading...")).toBeInTheDocument();

    // Unlike getByText, queryByText does NOT THROW an error when it can't find
    // something -- it returns null. queryBy* is used to verify that something
    // is NOT ON THE SCREEN.
    expect(screen.queryByText("Data loaded successfully.")).not.toBeInTheDocument();
  });

  it("shows an error message", () => {
    render(<StatusMessage status="error" />);
    expect(screen.getByText("Something went wrong.")).toBeInTheDocument();
  });

  it("shows the success message by default", () => {
    render(<StatusMessage status="success" />);
    expect(screen.getByText("Data loaded successfully.")).toBeInTheDocument();
  });
});
```

Üç ayrı `it` bloğu, `status` prop'unun üç farklı değeri için component'i ayrı ayrı render edip doğru mesajın göründüğünü kontrol ediyor. İlk testte ayrıca `queryByText` kullanılıyor: `getByText`'in aksine, eleman bulunamazsa hata FIRLATMAZ, `null` döner -- bu yüzden bir şeyin EKRANDA OLMADIĞINI doğrulamak için `getByText` değil `queryByText` kullanılır.

Özet ve Terimler Sözlüğü

Vitest testleri ÇALIŞTIRIR, React Testing Library component'leri sahte bir DOM'a yerleştirip SORGULAMAMIZI sağlar. `render()` bir component'i DOM'a yerleştirir; `screen` o DOM'u sorgulamak için kullanılır. `getByRole` / `getByLabelText` / `getByText`, bir eleman bulamazsa hata fırlatır; `queryBy*` varyantları bulamazsa `null` döner ve bir şeyin EKRANDA OLMADIĞINI doğrulamak için kullanılır. `@testing-library/jest-dom`, `toBeInTheDocument()` gibi DOM'a özel matcher'lar ekler.

Terimler Sözlüğü

Test Runner — Testleri bulup çalıştıran, sonuçları raporlayan araç (Vitest).

jsdom — Node içinde çalışan, gerçek bir tarayıcıyı SİMÜLE eden sahte bir DOM ortamı.

Matcher — `expect(...)`'ten sonra zincirlenen, belirli bir koşulu doğrulayan fonksiyon (`toBeInTheDocument()`, `toHaveValue()` gibi).