JEST 튜토리얼

정보통신공학부 2018036044 이승준

목차

JEST가 무엇인가?

"JEST"는 주로 JavaScript 프로젝트에서 사용되는 테스트 프레임워크입니다. JEST는 Facebook에서 개발하고 유지 보수하며, JavaScript 및 TypeScript를 사용하는 프로젝트에서 유용하게 활용됩니다. 주로 React 애플리케이션을 테스트하는 데 많이 사용되지만, 다른 종류의 JavaScript 프로젝트에서도 사용할 수 있습니다.

JEST는 다음과 같은 특징을 갖고 있습니다:

1. **쉬운 설정**: JEST는 기본 설정이 간단하며, 대부분의 경우 추가적인 설정이 필요하지 않습니다.

2. **자동 mocking**: 함수나 모듈의 mocking을 자동으로 처리해주어, 모듈 간의 의존성을 쉽게 관리할 수 있습니다.

3. **Snapshot 테스팅**: UI 컴포넌트의 렌더링 결과물을 저장하고, 이후에는 이와 비교하여 변경 여부를 감지하는 방식으로 UI 테스트를 할 수 있습니다.

4. **비동기 코드 테스트**: 비동기 코드에 대한 테스트를 간편하게 작성할 수 있도록 도와줍니다.

5**. Coverage 보고서 생성**: 코드 커버리지를 확인하고 테스트되지 않은 부분을 식별할 수 있는 보고서를 생성할 수 있습니다.

프로젝트의 테스트 환경을 설정하고, 유닛 테스트 및 통합 테스트를 쉽게 작성할 수 있도록 도와주는 JEST는 JavaScript 개발자들 사이에서 인기 있는 테스트 도구 중 하나입니다.

React가 무엇인가?

React는 Facebook에서 개발한 사용자 인터페이스(UI) 라이브러리로, 단일 페이지 애플리케이션(SPA) 및 모바일 애플리케이션의 사용자 인터페이스를 구축하는 데 사용됩니다. React는 선언적이고 효율적인 UI를 만들기 위한 컴포넌트 기반 아키텍처를 제공하며, 가상 DOM(Virtual DOM)을 사용하여 성능을 최적화합니다.

React는 Facebook에서 개발한 사용자 인터페이스(UI) 라이브러리로, 단일 페이지 애플리케이션(SPA) 및 모바일 애플리케이션의 사용자 인터페이스를 구축하는 데 사용됩니다. React는 선언적이고 효율적인 UI를 만들기 위한 컴포넌트 기반 아키텍처를 제공하며, 가상 DOM(Virtual DOM)을 사용하여 성능을 최적화합니다.

React의 주요 특징과 개념은 다음과 같습니다:

1. 컴포넌트 기반:

- React 애플리케이션은 여러 개의 독립적인 컴포넌트로 구성됩니다. 각 컴포넌트는 자체적으로 상태(state)와 속성(props)을 가지며, 재사용성과 유지보수가 쉽도록 설계되어 있습니다.

2. 가상 DOM:

- React는 가상 DOM을 사용하여 실제 DOM 조작을 최소화하고 성능을 향상시킵니다. 가상 DOM은 메모리에 존재하는 가벼운 복제품으로, 실제 DOM과의 변경 사항을 비교하고 효율적으로 업데이트합니다.

3. JSX (JavaScript XML):

- React에서는 JSX라는 문법을 사용하여 JavaScript 코드 안에 XML과 유사한 구문을 작성할 수 있습니다. JSX는 가독성이 높고 컴포넌트의 구조를 직관적으로 나타내는 데 도움이 됩니다.

4. 단방향 데이터 바인딩:

- React는 단방향 데이터 바인딩을 통해 데이터의 흐름을 쉽게 관리합니다. 부모 컴포넌트에서 자식 컴포넌트로 데이터를 전달하고, 상태 변화에 따라 UI가 자동으로 업데이트됩니다.

5. 상태 관리:

- React 컴포넌트는 상태(state)를 가질 수 있습니다. 상태는 컴포넌트의 동적인 데이터를 나

타내며, `setState` 메서드를 사용하여 업데이트됩니다.

6. 라이프사이클 메서드:

- React 컴포넌트는 라이프사이클 메서드를 통해 컴포넌트의 생성, 업데이트, 소멸과 같은 단계에서 특정 동작을 수행할 수 있습니다.

7. 커뮤니티와 생태계:

- React는 활발한 개발자 커뮤니티와 다양한 라이브러리, 도구, 프레임워크를 풍부하게 갖추고 있습니다. 이로 인해 React 기반의 다양한 프로젝트를 쉽게 구축할 수 있습니다.

React는 주로 웹 애플리케이션 개발에서 사용되지만, React Native라는 독립적인 프로젝트를 통해 네이티브 모바일 애플리케이션도 개발할 수 있습니다.

Mocking이 무엇인가?

모킹(Mocking)은 소프트웨어 개발에서 사용되는 테스트 기법 중 하나입니다. 모킹은 특정 객체나 모듈을 대체하여 테스트하는 방법으로, 주로 유닛 테스트에서 자주 활용됩니다.

1. 목적:

- 외부 의존성이 있는 코드를 테스트할 때, 해당 의존성을 실제로 호출하는 대신 가짜(mock) 객체로 대체함으로써 독립적인 테스트를 수행할 수 있습니다.

- 모킹은 주로 외부 API 호출, 데이터베이스 액세스, 파일 시스템 액세스 등과 같은 리소스에 의존하는 코드를 테스트할 때 사용됩니다.

2. 장점:

- 테스트 속도를 높일 수 있습니다. 실제 외부 리소스를 호출하는 것은 느릴 수 있지만, 가짜(mock) 객체를 사용하면 빠르게 테스트를 실행할 수 있습니다.

- 테스트 간의 독립성을 유지합니다. 한 테스트가 다른 테스트에 영향을 주지 않도록 하기 위해 외부 의존성을 모킹할 수 있습니다.

3. 예시:

- 예를 들어, 데이터베이스에 대한 테스트를 수행할 때, 실제로 데이터베이스에 연결하지 않고 데이터베이스를 사용하는 코드를 가짜(mock) 데이터베이스 객체로 대체하여 테스트를 수행할 수 있습니다.

```
// 원래의 코드
class Database {
  save(data) {
    // 실제로 데이터베이스에 데이터 저장
  }
}

// 모킹된 코드
class MockDatabase {
  save(data) {
    // 가짜 데이터베이스에 데이터 저장 (실제로는 저장되지 않음)
  }
}

// 테스트 코드
test('save data to database', () => {
  const mockDB = new MockDatabase();
  const data = { key: 'value' };

  // 모킹된 데이터베이스 객체를 사용하여 테스트
  const result = saveDataToDatabase(mockDB, data);

  // 테스트 결과 검증
  expect(result).toEqual(expectedResult);
});
```

위의 예시에서 `saveDataToDatabase` 함수는 실제 `Database` 객체 대신 `MockDatabase` 객체를 사용하여 데이터를 저장하고 테스트를 수행합니다.

JEST 사용방법

1. Jest 설치

프로젝트 디렉토리에서 다음 명령어를 사용하여 Jest를 설치합니다.

`npm install --save-dev jest` 또는 `yarn add --dev jest`

2. 테스트 파일 작성

Jest는 test 또는 spec으로 끝나는 파일을 자동으로 찾아 테스트합니다. 예를 들어, sum.js 파일이 있고 그에 대응하는 테스트 파일 sum.test.js를 작성합니다.

```
// sum.js
function sum(a, b) {
  return a + b;
}
module.exports = sum;
```

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

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

3. 테스트 실행

package.json 파일에 테스트 스크립트를 추가하고, 다음과 같이 Jest를 실행합니다.

```
// package.json
{
  "scripts": {
    "test": "jest"
  }
}
```

터미널에서 다음 명령어를 실행하여 테스트를 실행합니다.

`npm test` 또는 `yarn test`

Jest 주요 개념과 함수

test(name, fn, timeout): 테스트를 정의하는 함수. name은 테스트의 이름을 나타내고, fn은 실제 테스트 코드를 담고 있는 함수입니다.

expect(value): 테스트에서 예상하는 값을 나타내는 함수. Jest는 이를 사용하여 다양한 매처(matcher) 함수를 제공하며, 특정 조건을 검증할 수 있습니다.

```
// 예시
test('2 + 2는 40이다.', () => {
  expect(2 + 2).toBe(4);
});
```

기타 주요 함수:

- **toBe**(expected): 값이 정확하게 일치하는지 확인.
- **toEqual**(expected): 객체 또는 배열의 값을 재귀적으로 확인.

- **not.toBe**(expected): 값이 정확하게 일치하지 않는지 확인.

그 외에도 다양한 matcher 함수가 있습니다.

이외에도 Jest는 Snapshot 테스팅, 비동기 코드 테스트, 모킹 등 다양한 기능을 제공합니다. Jest의 자세한 사용법은 Jest 공식 문서(https://jestjs.io/)를 참고하시면 도움이 될 것입니다.