

15. Улучшаем опыт тестирования с Jest

Содержание урока

- Обзор;
- Знакомство с Jest;
- Юнит-тестирование с Jest;
- Snapshot-тестирование;
- Подведём итоги.

Обзор

Привет! 🙌🎓 В этой части конспекта мы познакомимся с `Jest` — мощным фреймворком для тестирования JavaScript. 🚀

К делу! 🦊

Знакомство с Jest

Эффективное тестирование программного кода, как и многие другие процессы в программировании, предусматривает присутствие определенного инструментария. 🛠️

К счастью, в современном JavaScript-комьюнити имеется отличнейший фреймворк для работы с тестами — Jest.

[Jest](#) — это JavaScript-фреймворк, разработанный и поддерживаемый компанией Facebook.

Релиз Jest в open source состоялся в 2014 году.

Jest совмещает в себе:


- Тест-раннер (англ. `"test runner"`);
- Библиотеку ассертов (англ. `"assertion library"`);
- Библиотеку моков (англ. `"mock library"`);
- и еще несколько инструментов.

С учётом обширного инструментария, Jest именуется фреймворком, а не библиотекой. 🖼️

К преимуществам Jest можно отнести следующие качества:


- Быстродействие — Jest запускает тесты в параллельном режиме, что увеличивает производительность;


- Наличие встроенных ассертов и моков — можно мокать «сайд-эффекты» и модули;
- Snapshot-тестирование — уникальная фишка Jest, позволяющая писать тесты в невероятно лаконичной форме. Мы рассмотрим данную технику в следующей секции этого урока;
- Репорт уровня покрытия кода — для получения полноценного и красиво оформленного test-coverage достаточно всего лишь включить несколько флагов;
- Запуск тестов в изолированной песочнице — каждый тест запускается в новом, чистом окружении, не пересекаясь с другими тестами, повышая тем самым общий уровень качества тест-сессий;
- Простая интеграция с другими инструментами для тестирования — работает хорошо почти с любой другой библиотекой или фреймворком.

 Хозяйке на заметку:

Jest идет вместе с `JSDOM`, JavaScript-имплементаций DOM-окружения, что открывает возможность легко тестировать модули, предназначенные для работы в окружении браузера, однако все тесты запускаются в окружении `Node.js`, что открывает дополнительные возможности.

Юнит-тестирование с Jest

Jest делает процесс юнит-тестирования лёгким и приятным.  Давай попробуем протестировать функцию, принимающую `два числовых аргумента` и в результате возвращающую их `сумму`. Мы будем следовать подходу `TDD` в разработке данной функции и сперва напишем тест.

 Хозяйке на заметку:

По умолчанию Jest ищет и выполняет тесты, найденные в папках с именем `__tests__`. Альтернативно можно называть файлы с тестами в формате `*.test.js` или `*.spec.js`.

 Пример кода 15.1:

```
1 // sum.test.js
2 import { sum } from './sum';
3
4 test('sum function should add two numbers passed as arguments correctly',
5   () => {
6     expect(sum(1, 2)).toBe(3);
7   });
```

С учётом написанного теста, мы можем запустить команду `npm test`, после чего Jest сообщит в терминале о неуспешно прошедшем тесте, что ожидаемо, ведь функция `sum` еще не имплементирована:

```

1  FAIL   development/example/sum.test.js
2    • Adds 1 + 2 to equal 3
3
4    TypeError: (0 , _sum2.default) is not a function
5
6  Test Suites: 1 failed, 1 total
7  Tests:      2 failed, 2 total
8  Snapshots:  0 total
9  Time:       1.156s
10 Ran all test suites.

```

Имплементация функции `sum`:

```

1  // sum.js
2  export const sum = (a, b) => {
3
4    return a + b;
5  }

```


Теперь, запустив команду `npm test` во второй раз, Jest сообщит следующую информацию в терминале:

- Тесты были выполнены успешно;
- Список выполненных тестов и текст описания каждого теста;
- Количество успешно выполненных «наборов тестов» (англ. `"test suite"`);
- Количество успешно выполненных тестов;
- Количество протестированных snapshots;
- Общее время выполнения всех тестов.

```

1  PASS   development/example/sum.test.js
2    ✓ Adds 1 + 2 to equal 3 (2ms)
3
4  Test Suites: 1 passed, 1 total
5  Tests:      1 passed, 1 total
6  Snapshots:  0 total
7  Time:       0.857s, estimated 1s
8  Ran all test suites.

```

 Хозяйке на заметку:

Разница между `Test Suites` и `Tests` в том, что `test suite` — это файл с группой тестов, а `test` — это единичный тест, самая атомарная единица.

Имплементация теста для функции `sum` в примере кода 15.1 на строке кода 5 описывает использование ассерта `toBe`, сравнивающего два примитива строгим ссылочным сравнением JavaScript: `===`.

В программировании термин `ассерт` означает предикат, размещённый в программе и указывающий на то, что разработчик считает этот предикат в этом месте программы всегда истинным. А `предикат` — это утверждение, высказанное о субъекте.

Jest предоставляет ещё много полезных методов-ассертов для тестирования различных значений. Например, для тестирования возвращаемого значения в виде объекта ассерт `toBe` не подойдёт, потому что в основе его реализации лежит `===`, строгое ссылочное сравнение. Это комплементирует со ссылочной природой объектов в JavaScript. Для сравнения объектов необходим другой подход — сравнение по `значению`, а не по `ссылке`. Для такого случая существует ассерт `toEqual`, который безопасно сравнивает два объекта по `значению`.

 Хозяйке на заметку:

В арсенале Jest есть ещё много полезных ассертов. Ты можешь ознакомиться с ними поподробнее на официальном сайте с [документацией](#). 💡

Snapshot-тестирование

`Snapshot-тестирование` (`"snapshot"` означает `снимок`) — это уникальная фишка Jest, сериализирующая результат вызова тестируемой сущности и сохраняющая полученный результат в JSON-файл с расширением `.snap`, помещая его в директорию `__snapshots__` рядом с файлом с тестом. При последующих тестах такой сущности результат ее вызова сериализируется снова и сравнивается с имеющимся snapshot. На основе последнего сравнения делается вывод — тест пройден или нет. ✅

Например, чтобы протестировать уже знакомую нам функцию `sum`, следуя подходу `snapshot-тестирования`, необходима следующая инструкция.

 Пример кода 15.2:

```
1 // sum.test.js
2 import { sum } from './sum';
3
4 test('sum function should add two numbers passed as arguments correctly',
  () => {
5   expect(sum(1, 2)).toMatchSnapshot();
6 });
```

В `примере кода 15.2` ассерт `toMatchSnapshot` Jest сериализирует результат вызова функции `sum(1, 2)` в JSON-объект и сформирует файл с именем `sum.test.js.snap`, имеющий следующий вид:

```
1 // Jest Snapshot v1, https://goo.gl/fbAQLP
2
3 exports[`sum function should add two numbers passed as arguments
  correctly 1`] = `3`;
```

Последующие перезапуски этого теста будут сравниваться с данным snapshot. Если по какой-то причине результат вызова `sum` в данном тесте вернёт значение, отличное от `3`, Jest сообщит о несходстве в терминале разработчика. Это очень удобно. 📖

Подведём итоги

В этом уроке мы ознакомились с мощным фреймворком для тестирования JavaScript-кода — Jest.

Jest предоставляет широкий спектр инструментов для тестирования и обладает очень удобной фичей snapshot-тестирования, с помощью которой можно писать тесты кратко и лаконично.

Спасибо, что остаёшься с нами! 🙌 В следующей части конспекта мы рассмотрим утилиту, с помощью которой можно тестировать React-компоненты. До встречи! 🍷

Мы будем очень признательны, если ты оставишь свой фидбек в отношении этой части конспекта на нашу электропочту `hello@lectrum.io`.