
OTUS
ОНЛАЙН-ОБРАЗОВАНИЕ
Онлайн-образование

Меня хорошо видно && слышно?

Ставьте + , если все хорошо
Напишите в чат, если есть проблемы

Unit тесты и data driven testing

Тимофеев Юрий

Правила вебинара

Активно участвуем

Задаем вопрос в чат или голосом

Off-topic обсуждаем в Slack #канал группы или #general

Вопросы вижу в чате, могу ответить не сразу

Программа вебинара

Виды тестирования

Инструменты тестирования

Практика *

Итоги

* Самостоятельная работа: приготовление

Вам потребуется:
- git,
- npm
- ваш любимый редактор кода

Проверьте, что у вас свежая версия node и npm. Взять можно здесь: https://nodejs.org/en/download/

Цели вебинара | После занятия вы сможете

1. Рассказать про виды и инструменты тестирования
2. Рассказать про принцип AAA
3. Написать Unit-тесты

Входное тестирование:

Пожалуйста, пройдите установочный тест, ссылка в чате

Срок: 5 минут

Вопрос:

Приходилось ли заниматься тестированием, какие инструменты использовали?

Тестирование в эру old school: waterfall

Тестирование сейчас: SDLC (Software Development Life Cycle)

Пирамида тестирования

Unit тестирование

Многочисленные и быстрые

Без внешних зависимостей

White boxed

Интеграционное/компонентное тестирование

Проверка взаимодействия

Соответствие спецификации

Black boxed

Сквозное / E2E тестирование

Имитация пользователя

Сложные и медленные

Black boxed

Инструменты тестирования

Test runner

Assertion library

Mocking library

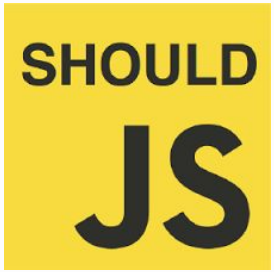
Coverage reporter

Test runner: среда выполнения тестов

- Организация
- Запуск тестов
- Проверка
- Отчет

Assertion library: проверка предположений

- Проверка предположений
- Падение тестов



Mocking library: имитация зависимостей

- Mock - настраиваемая имитация поведения
- Stub - жесткий объект-заглушка
- Spy - умеет протоколировать

Coverage reporter: отчет о покрытии кода тестами

Проверяет, какие участки кода были запущены при тестировании и строит отчет

Обоснованный выбор

Jest

- Test runner
- Assertion library
- Mocking library
- Coverage reporter

Вопрос: что здесь происходит?

```
describe('Trivial test', () => {
  test('2*2=4', () => {
    expect(2*2).toBe(4)
  })
})
```

Jest: соглашения по файловой структуре

```
/project-folder
  /.git
  - package.json
  - jest.config.js
  /src
    - component1.js
    - component2.js
   ...
  /specs
    - component1.spec.js
    - component2.spec.js
   ...
```

Jest: именование файлов

Имена файлов тестов

JS-проекты

`<component>.spec.js`
`<component>.test.js`

TS-проекты

`<component>.spec.ts`
`<component>.test.ts`

Jest: describe - структуризация

```
describe(name, fn)
// https://jestjs.io/docs/en/api#describename-fn
```

```
describe('Component-to-test', () => {
    ..............
    describe('Feature-to-test', () => {
        ..............
    })
    ..............
})
```

Jest: test/it - тесты

```
test(name, fn, timeout)
it(name, fn, timeout)
// https://jestjs.io/docs/en/api#testname-fn-timeout
```

```
describe('Component-to-test', () => {
  it('Loads without error', () => {
    ...........
  })
  it('Really works', () => {
    ...........
  })
})
```

Jest: expect - проверки (Assert)

```
expect(expression)
   .toBe(value)
   .toBeDefined()
   .toHaveLength(len)
// https://jestjs.io/docs/en/expect
```

```
describe('Component-to-test', () => {
  it('Reads data', () => {
     .........
     expect(reader.data).toHaveProperty('status')
  })
})
```

Jest: параметризация

```
describe.each(table)(name, fn, timeout)

describe.each`table`(name, fn, timeout)

// https://jestjs.io/docs/en/api#describeeachtablename-fn-timeout
```

```
test.each(table)(name, fn, timeout)

test.each`table`(name, fn, timeout)

// https://jestjs.io/docs/en/api#testeachtablename-fn-timeout
```

Jest: пример параметризации

```
test.each`
  a    | b    | expected
  ${1} | ${1} | ${2}
  ${1} | ${2} | ${3}
  ${2} | ${1} | ${3}
`('returns $expected when $a is added $b',
   ({a, b, expected}) => {
      expect(a + b).toBe(expected);
   }
 );
```

Декларация
Значения

Шаблон
Переменные
Работа

https://jestjs.io/docs/en/api#testeachtablename-fn-timeout

Jest: хуки

beforeEach / afterEach / beforeAll / afterAll
// https://jestjs.io/docs/en/api

```
describe('Component-to-test', () => {
  let objectToTest
  beforeEach(() => {
    objectToTest = new Component()
    objectToTest.setup()
  })
  ....................
})
```

Arrange Act Assert

```
const customer = new
Customer()
const ticket  = new Ticket()

customer.purchase(ticket)

expect(customer)
.toHaveProperty('ticket')
```

Arrange

Act

Assert

Livecoding:

Тестирование функции max


~5 минут
Livecoding:
Тестирование функции factorial

Самостоятельная работа:

~10 минут

- Откройте ссылку в чате и сделайте форк проекта
- Откройте /specs/factorial.spec.js
- Напишите не менее 5 тестов для функции factorial. Проверьте правильные и неправильные параметры
- Пришлите в чат ссылку

Подсказка:
https://gist.github.com/georgius1024/932b421155c873461eb816f3a9c86173

Тестирование React-компонентов

1) Нужны библиотеки:
- enzime
- @testinglibrary/react
- react-test-renderer

2) Разработчики должны добавлять атрибут **data-testid**

Пример теста

```
const c = render('<MyComponent/>')              // Arrange

c.getByTestId('input').value = 'My name'        // Act
c.getByTestId('button').click()                 // Act

expect(c.getByTestId('result')).toBeDefined() // Assert
```


~5 минут
Livecoding:
Тестирование React-компонента

Самостоятельная работа:

~10 минут

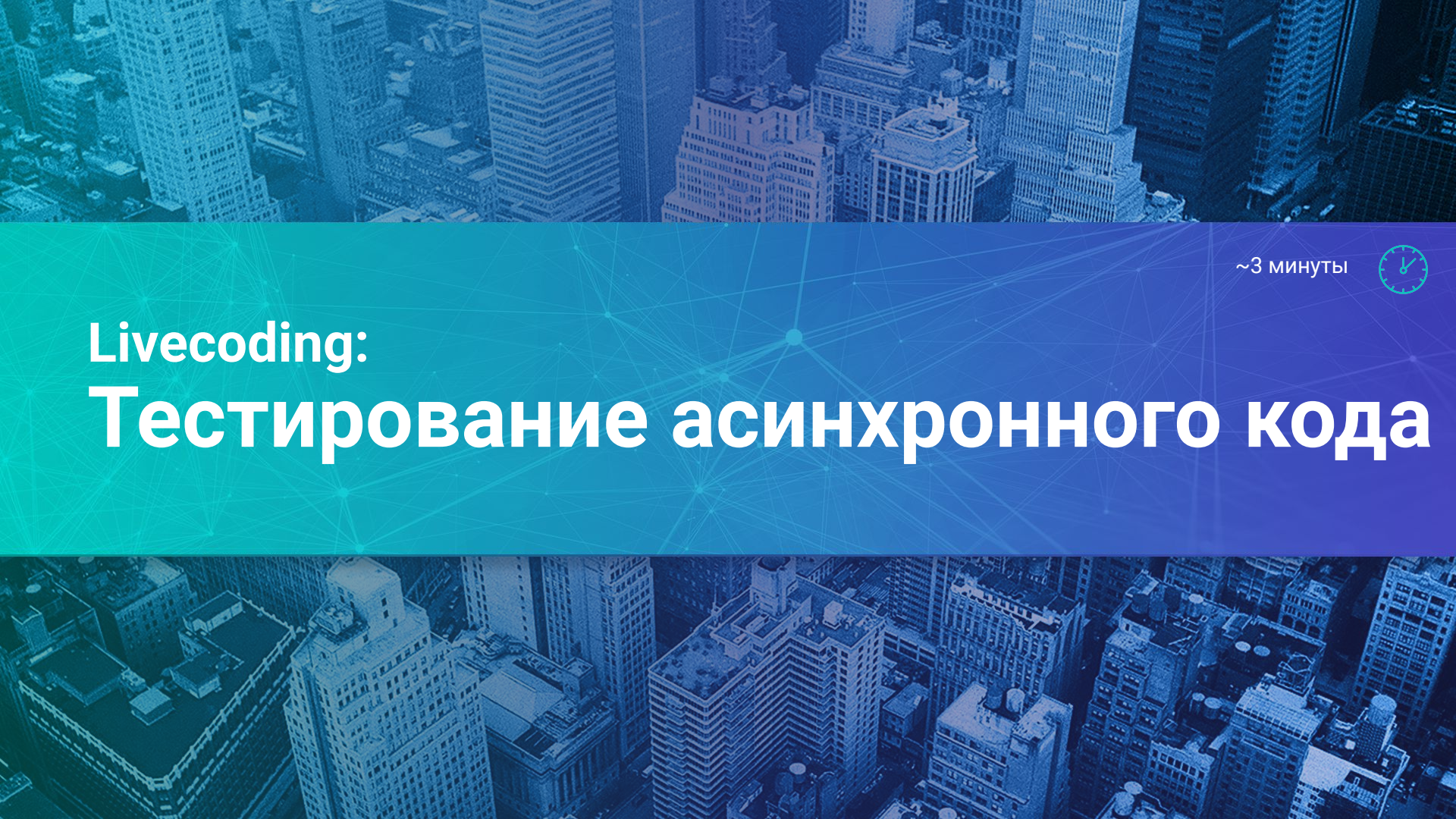
- Откройте ссылку в чате и сделайте форк проекта
- Откройте /specs/Calc.spec.js
- Добавьте тесты для всех 4-х арифметических операций
- Добавьте тест для кнопки очистки
- Пришлите в чат ссылку

Подсказка:
https://gist.github.com/georgius1024/932b421155c873461eb816f3a9c86173

Тестирование асинхронного кода: async / await

Пример теста

```
import searchService from '../src/services/search' // Arrange

it('return search results', async () => {
  const result = await searchService.find('test')  // Act
  expect(result).toHaveProperty('entries')         // Assert
})
```


~3 минуты
Livecoding:
Тестирование асинхронного кода

Покрытие кода тестами

Тестируемый код

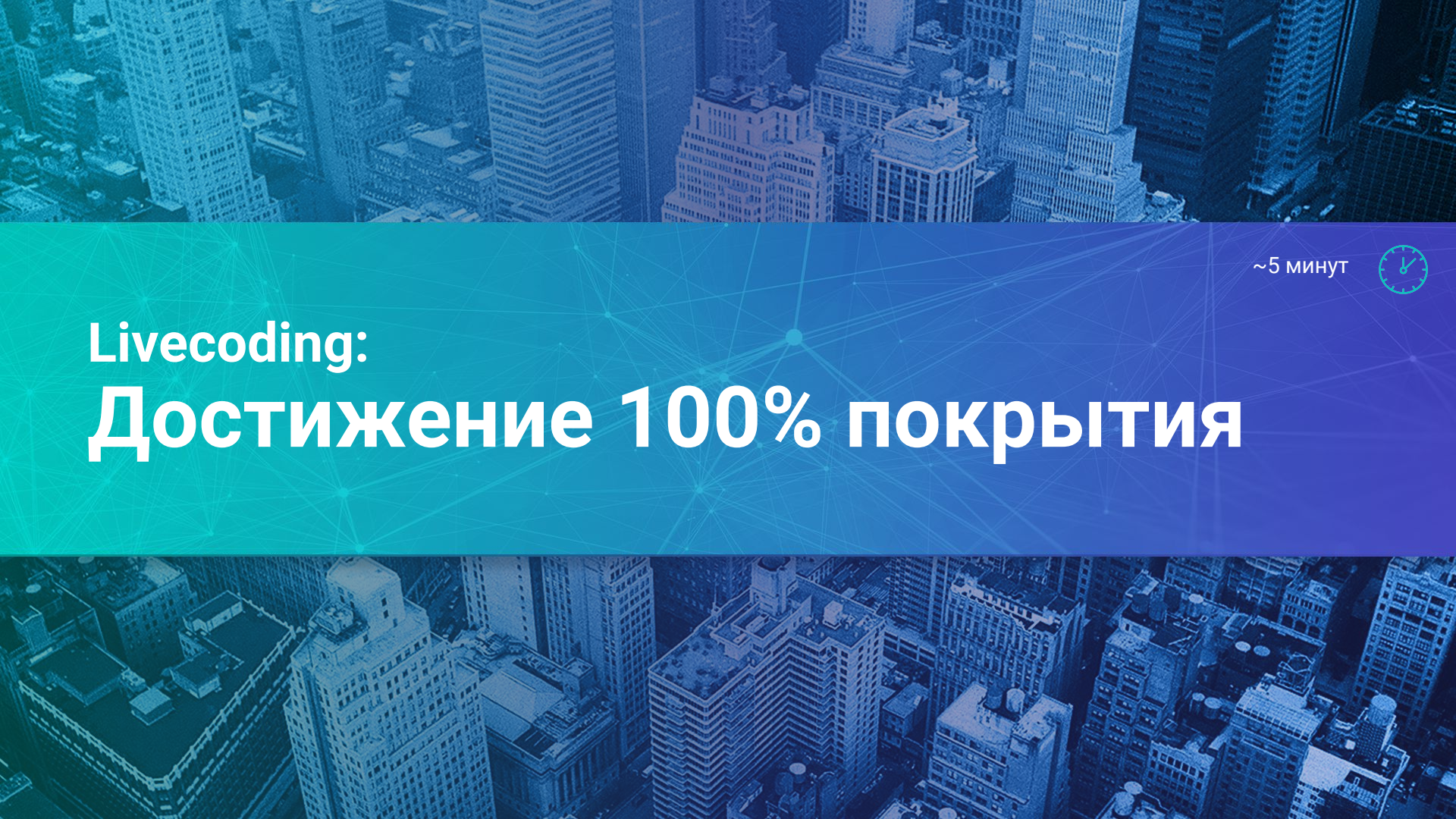
```
function func1(x) {....} // Covered

function func2(x) {....} // Not covered

function func3(x) {....} // Not covered
```

Пример теста

```
it('func 1 return correct value', () => {
  expect(func1(x).toBe(CorrectValue)
})
```

Метрики покрытия тестами

- Function coverage
- Statement coverage
- Edge coverage
- Branch coverage
- Condition coverage


~5 минут
Livecoding:
Достижение 100% покрытия

Самостоятельная работа: приготовление

Вам потребуется git, npm и ваш любимый редактор кода

0) Проверьте, что у вас свежая версия node и npm
Взять можно здесь: https://nodejs.org/en/download/

1) Ссылка на репозиторий в чате

2) git clone https://github.com/georgius1024/calc-test

3) cd calc-test

4) npm install

Jest: параметризация, повтор

```
test.each`
   a       | op      | b       | expected
   ${2}    | ${'*'} | ${2}    | ${4}
   ${2}    | ${'&'} | ${2}    | ${'error'}
 `('$a $op $b = $expected', ({ a, op, b, expected }) => {
   if (expected === 'error') {
     expect(() => calculator(a, op, b)).toThrow();
   } else {
     expect(calculator(a, op, b)).toBe(expected);
   }
 });
```

Самостоятельная работа:

~10 минут

- Ознакомьтесь с src/calculator.js
- Откройте specs/calculator.spec.js
- Добавьте проверки для всех 4-х арифметических операций
- Добавьте проверки для ошибочных параметров и доведите покрытие до 100%
- Пришлите в чат ссылку

Подсказка:
https://gist.github.com/georgius1024/2b5ba1be17c3b00110cf6853ec87d740

Пройдите тестирование по ссылке в чате

Срок: 5 минут

Что дальше?

Материалы:

https://habr.com/ru/post/358950/ Пирамида тестирования

https://habr.com/ru/post/507594/ Анатомия юнит тестирования

Руководства:

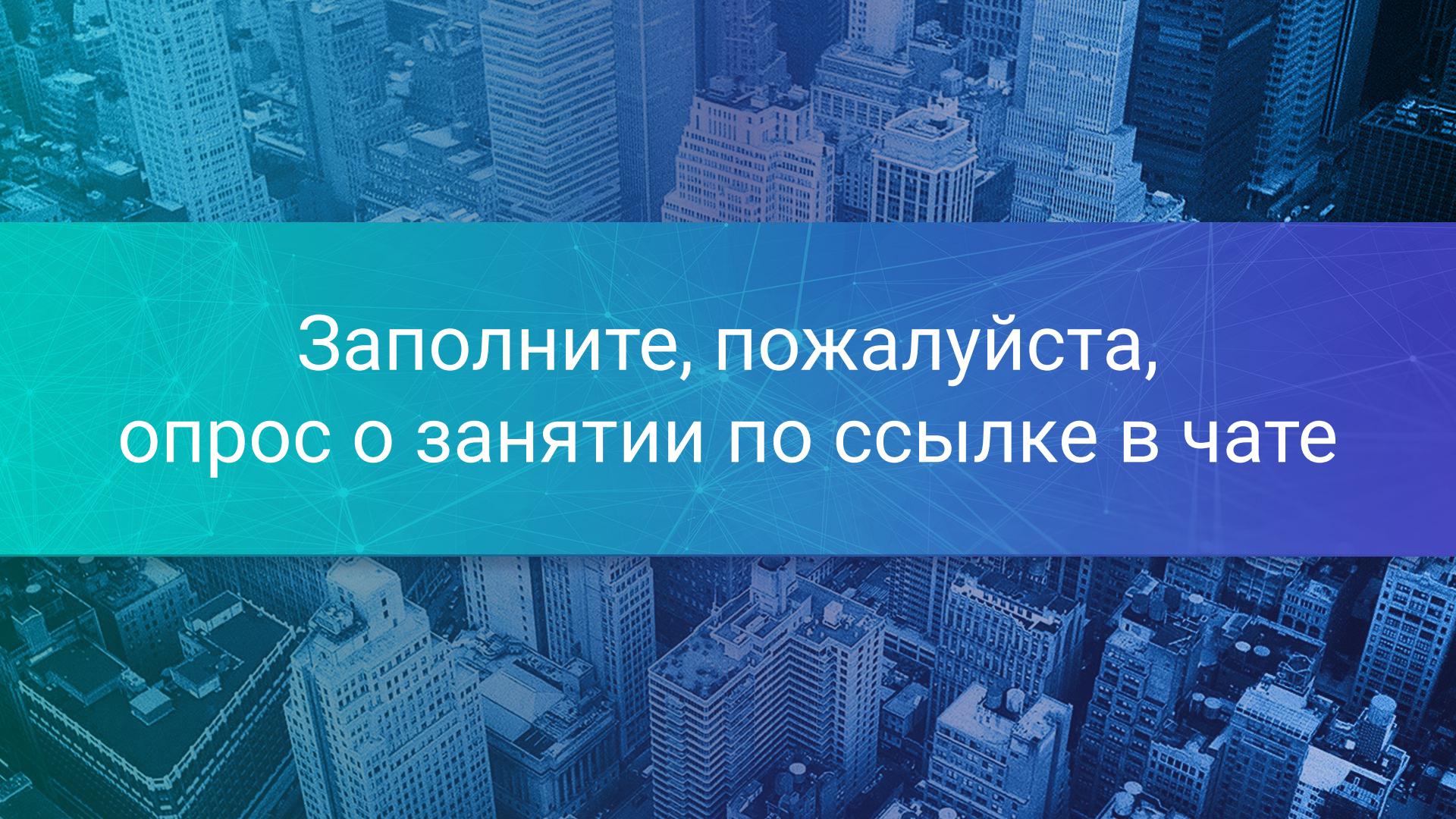
https://jestjs.io/docs/en/api глобальные объекты

https://jestjs.io/docs/en/expect проверки

https://jestjs.io/docs/en/asynchronous тестирование асинхронного кода

Вопрос:

Чему мы сегодня научились?


Заполните, пожалуйста,
опрос о занятии по ссылке в чате


Спасибо за внимание!
Приходите на следующие вебинары