# JEST

INFOSHARE ACADEMY
23RD NOVEMBER 2019

# WHO AM I?

Maciej Zbierowski

Software Engineer in Ensono.

Huge movie fan, cat lady owner :D

# PLAN FOR CLASSES ☺

**Presenting**

**Live coding**

**Questions I hope :D**

15:00 – 16:30

16:30 – 17:00 – break

17:00 – 18:30

18:30 – 19:00 – break

19:00 – 21:00

# PLAN FOR PRESENTATION

1. UNIT TESTING
2. JEST – BASIC INFORMATION
3. MOCKING – GENERAL IDEA
4. MOCKING IN JEST
5. SAMPLE APP

Explain node js module.exports / require

Może zadanie na sort? Usuwanie powtórek?

# BEFORE GETTING INTO JEST …

We should know:

- Javascript
- Node JS – **require**, **module.exports**, **exports, package.json, npm run**

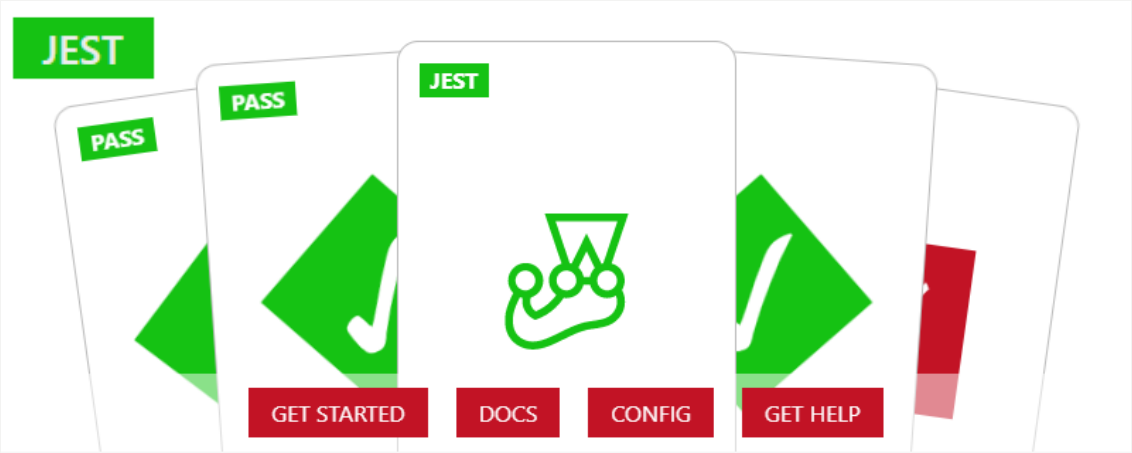
# UNIT TESTING

Unit – the smallest testable part of any software.

Software testing where individual units (components) of a software are tested.

Purpose: to validate that each unit of the software performs as designed.

# JEST




Test Framework with focus on simplicity

Created by Facebook engineers for its React project

https://jestjs.io/en/

# JEST SYNTAX

```
test('two plus two is four', () => {
  expect(2 + 2).toBe(4);
});
```

```
beforeEach(() => {
  initializeCityDatabase();
});

afterEach(() => {
  clearCityDatabase();
});

test('city database has Vienna', () => {
  expect(isCity('Vienna')).toBeTruthy();
});

test('city database has San Juan', () => {
  expect(isCity('San Juan')).toBeTruthy();
});
```

```
describe('matching cities to foods', () => {
  // Applies only to tests in this describe block
  beforeEach(() => {
    return initializeFoodDatabase();
  });

  test('Vienna <3 sausage', () => {
    expect(isValidCityFoodPair('Vienna', 'Wiener Schnitzel')).toBe(true);
  });

  test('San Juan <3 plantains', () => {
    expect(isValidCityFoodPair('San Juan', 'Mofongo')).toBe(true);
  });
});
```

# JEST CONFIGURATION - NPM

To use JEST we need to install it with npm or yarn:

***npm install --save-dev jest***

We need to change package.json:

```
{
  ...
  "test": "jest"
}
```

We could use different JEST options:

- Run only tests you want: **jest path/to/my-test.js**
- Run with **watch** mode: **jest --watch**
- Run tests with coverage information: **jest --coverage**

# MATCHERS - BASIC

**expect().toBe() –** for primitive types

**expect().toEqual() –** for reference types (objects

# MATCHERS - TRUTHINESS

expect().toBeNull

expect().toBeUndefined()

expect().toBeDefined

expect().toBeTruthy()

expect().toBeFalsy()

# MATCHERS - NUMBERS

**expect().toBeGraterThan()**

**expect().toBeGreaterOrEqual()**

**expect().toBeLessThan()**

**expect().toBeLessThanOrEqual()**

!!! For float numbers use **expect.toBeCloseTo()** (rounding error)

# MATCHERS – STRINGS AND ARRAYS

Strings: **expect().toMatch()**

Arrays: **expect().toContain()**

# TESTING ASYNCHRONOUS CODE

Promises

.resolves() .rejects()

Async/wait

# PROMISES

Return a promise from your test and JEST will wait for that promise to resolve. If the promise is rejected test will automatically fail.

!! Be sure to **return** the promise

```
test(" ...", () => {
   return promiseFunction().then ...
});
```

If you expect the promise to be rejected use the **.catch** method. Make sure to add **expect.assertions(1)** to verify that a certain number of assertions are called.

# RESOLVES/REJECTS AND ASYNC/AWAIT

Just a sugar for Promises ();

```
test(”to be 4”, () => {
  return expect(calculate(2,2)).resolves.toBe(4);
});

test(”to fetch with failures”, () => {
  return expect(calculate(2,2))).rejects.toMatch(”error”);
});
```

# MOCKING

Technique where code parts are replaced by dummy implementations that emulate real code. Mocking helps achieve isolation of tests.

Primarily used in unit testing

”Something made as an immitation”

# MOCKING

Primarily used in unit testing. An object under test may have dependencies on other (complex) objects. To isolate the behaviour of the object, you should replace the other objects by mocks that simulate the behaviour of the real objects. This is useful if the real objects are impractical to incorporate into the unit tests.

Mocking is creating objects that stimulate the behaviour of real objects.

# MOCKING

Mocking is creating objects that stimulate the behaviour of real objects.

Mocking is a technique to isolate test subjects by replacing dependencies with objects that you can control and inspect.

**The goal for mocking is to replace something we don't control with something we do, so it's important that what we replace it with, has all the features we need.**

# MOCKING IN JEST

A dependency can be anything your subject depends on, but it is typically a module that the subject imports.

When we talk about mocking in **JEST**, we're typically talking about replacing dependencies with the **MOCK FUNCTION**.

# MOCK FUNCTIONS

Allow you to test the links between code by erasing the actual implementation of a function, capturing calls to the function (and the parameters passed in those calls), capturing instances of contructor functions when instantiated with new, and allowing test-time configuration of return values.

Mock Function provides features to:

- Capture calls
- Set return values
- Change the implementation

The simplest way to create a Mock Function instance is with **jest.fn()**

# MOCK FUNCTIONS

There are three main types of module and function mocking in JEST:

- **jest.fn**: Mock a function
- **jest.mock:** Mock a module
- **jest.spyOn**: Spy or mock a function