Web Advanced

Jest

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Dep. PXL-IT – Elfde-Liniestraat 26 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Using Matchers

EDIT

Jest uses "matchers" to let you test values in different ways. This document will introduce some commonly used matchers. For the full list, see the `expect` API doc.

Common Matchers

The simplest way to test a value is with exact equality.

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

In this code, `expect(2 + 2)` returns an "expectation" object. You typically won't do much with these expectation objects except call matchers on them. In this code, `.toBe(4)` is the matcher. When Jest runs, it tracks all the failing matchers so that it can print out nice error messages for you.

`toBe` uses `Object.is` to test exact equality. If you want to check the value of an object, use `toEqual` instead:

```
test('object assignment', () => {
  const data = {one: 1};
  data['two'] = 2;
  expect(data).toEqual({one: 1, two: 2});
});
```

`toEqual` recursively checks every field of an object or array.

You can also test for the opposite of a matcher:

```
test('adding positive numbers is not zero', () => {
  for (let a = 1; a < 10; a++) {
    for (let b = 1; b < 10; b++) {
      expect(a + b).not.toBe(0);
    }
  }
});
```

https://jestjs.io/docs/using-matchers

`.toThrow(error?)`

Also under the alias: `.toThrowError(error?)`

Use `.toThrow` to test that a function throws when it is called. For example, if we want to test that `drinkFlavor('octopus')` throws, because octopus flavor is too disgusting to drink, we could write:

```
test('throws on octopus', () => {
  expect(() => {
    drinkFlavor('octopus');
  }).toThrow();
});
```

Note: You must wrap the code in a function, otherwise the error will not be caught and the assertion will fail.

```
import Point from '../../../src/js/drawing/Point';

test('constructor to generate a Point object',
    () => {
        let point = new Point(1,1);
        expect(point).toBeInstanceOf(Point);
    });

test('constructor to throw error if 1st param. not a number',
    () => {
        expect(() => {
            new Point("a", 1);
        }).toThrow(Error);
    });

test('constructor to throw error if 2nd parameter not a number',
    () => {
        expect(() => {
            new Point(1, "a");
        }).toThrow(Error);
    });
```

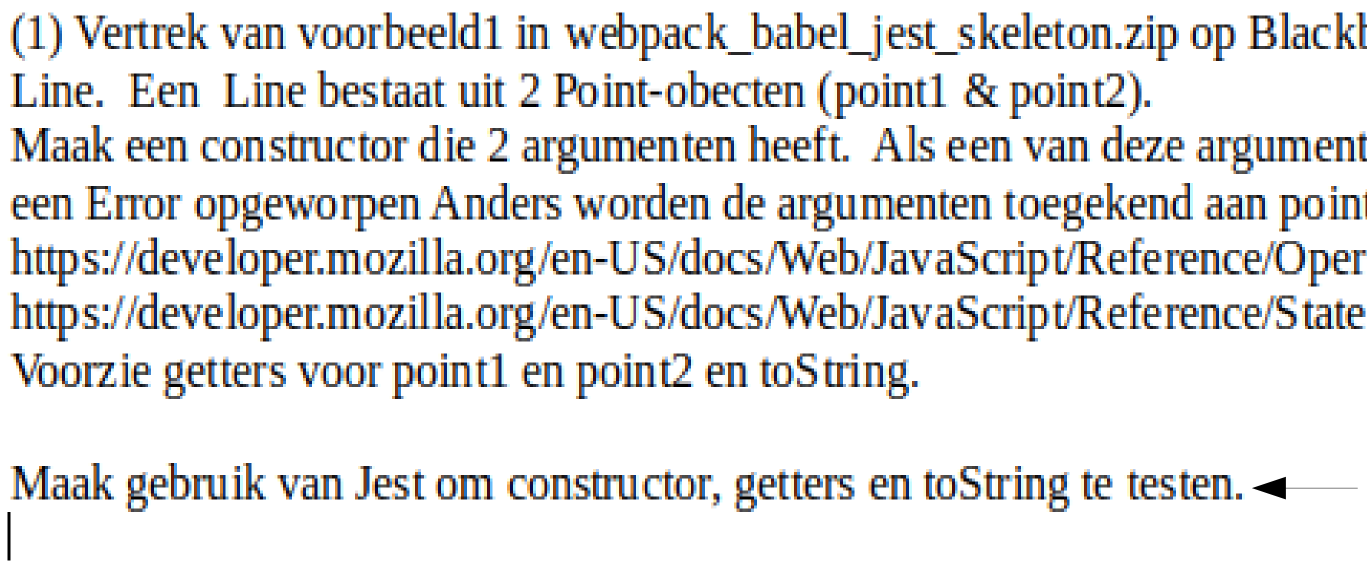
```
test('getX to return the correct value',
    () => {
        let point = new Point(1, 2);
        let x = point.x;
        expect(x).toBe(1);
    });

test('toString to return the correct value', () => {
    let point=new Point(1,2);
    let returnedString = point.toString();
    expect(returnedString).toBe("(1,2)");
});
```

describe: de 3 tests van de constructor worden samengevoegd

```
import Point from '../../../src/js/drawing/Point';
describe('constructor',
    () => {
        it('should generate a Point-object for valid args',
            () => {
                let point = new Point(1, 1);
                expect(point).toBeInstanceOf(Point)
            }
        )
        it('should throw error if 1st parameter is not a number',
            () => {
                expect(() => {
                    new Point("a", 1);
                }).toThrow(Error)
            })
        it('should throw error if 2d parameter is not a number',
           () => {
                expect(() => {
                    new Point(1, "a");
                }).toThrow(Error)
            })
    }
);
```

- `toBe` compares strict equality, using `===`
- `toEqual` compares the values of two variables. If it's an object or array, checks equality of all the properties or elements
- `toBeNull` is true when passing a null value
- `toBeDefined` is true when passing a defined value (opposite as above)
- `toBeUndefined` is true when passing an undefined value
- `toBeCloseTo` is used to compare floating values, avoid rounding errors
- `toBeTruthy` true if the value is considered true (like an `if` does)
- `toBeFalsy` true if the value is considered false (like an `if` does)
- `toBeGreaterThan` true if the result of expect() is higher than the argument
- `toBeGreaterThanOrEqual` true if the result of expect() is equal to the argument, or higher than the argument
- `toBeLessThan` true if the result of expect() is lower than the argument
- `toBeLessThanOrEqual` true if the result of expect() is equal to the argument, or lower than the argument
- `toMatch` is used to compare strings with regular expression pattern matching
- `toContain` is used in arrays, true if the expected array contains the argument in its elements set
- `toHaveLength(number)`: checks the length of an array
- `toHaveProperty(key, value)`: checks if an object has a property, and optionally checks its value
- `toThrow` checks if a function you pass throws an exception (in general) or a specific exception
- `toBeInstanceOf()`: checks if an object is an instance of a class

Oefening1



(1) Vertrek van voorbeeld1 in webpack_babel_jest_skeleton.zip op Blackboard. Maak de klasse Line. Een Line bestaat uit 2 Point-obecten (point1 & point2).
Maak een constructor die 2 argumenten heeft. Als een van deze argumenten geen Point is wordt een Error opgeworpen Anders worden de argumenten toegekend aan point1 en point2.
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/instanceof
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/throw
Voorzie getters voor point1 en point2 en toString.

Maak gebruik van Jest om constructor, getters en toString te testen.

Voorbeeld2

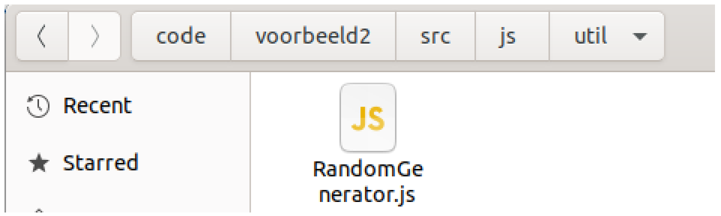
game/Game.js

- gooi N aantal dobbelstenen
- als alle dobbelstenen allemaal zijn: winst = N

- dependency:
  - util/Randomgenerator.js

Hoe Game testen die afhankelijk is van Randomgenerator?
-> dependency injection
-> mock van Randomgenerator

Voorbeeld2



```
"use strict";

export default class RandomGenerator{
    constructor(){}

    next(min, max){
        return Math.floor(min + ( max - min ) * Math.random());
    }
}
```

```
import RandomGenerator from '../util/RandomGenerator';
export default class Game{
    #randomGenerator;
    constructor( randomGenerator ){
        if (! (randomGenerator instanceof RandomGenerator)){
            throw new Error('Not a RandomGenerator');
        }
        this.#randomGenerator = randomGenerator;
    }

    throwDice( numberOfDice ){
        if (!Number.isInteger(numberOfDice)) {
            throw new Error('Not an integer');
        }
        if ( numberOfDice < 2 ) {
            return 0;
        }
        let firstThrow = this.#randomGenerator.next(1, 6);
        for (let i = 1 ; i < numberOfDice ; i++){
            let newThrow = this.#randomGenerator.next(1, 6);
            if( newThrow != firstThrow){
                return 0;
            }
        }
        return numberOfDice;
    }
}
```

d.i. constructor injection

```
const invalidArguments = [[1], [null], ["a"]];
describe('constructor',
    () => {
        it('should generate a Game-object if a valid argument is provided',
            () => {
                let randomGenerator = new RandomGenerator();
                let game = new Game(randomGenerator);
                expect(game).toBeInstanceOf(Game);
            }
        )
        test.each(invalidArguments)(
            'should throw an Error given an invalid argument',
            (argument) => {
                expect(() => {
                    let game = new Game(argument);
                }).toThrow(Error)
        });
        it('should throw error when 1st parameter is not provided',
            () => {
                expect(() => {
                    let game = new Game();
                }).toThrow(Error)
            })
    }
);
```

```
describe('throwDice',
     () => {
       it('should return 4 if 4 times the same value is thrown',
           () => {
             let randomGenerator = new RandomGenerator();
             let spy =
                 jest.spyOn(randomGenerator, 'next').mockImplementation(() => 1);
             let game = new Game(randomGenerator);
             let result = game.throwDice(4);
             expect(result).toBe(4);
             expect(spy).toHaveBeenCalled();
             spy.mockRestore();
           }
         )
```

Via jest.spyOn wordt een ‘neppe’ randomGenerator gemaakt (mock)
de methode next geef altijd waarde 1 terug

```
it('should return 0 if two dice throws are not equal',
      () => {
        let randomGenerator = new RandomGenerator();
        let spy = jest.spyOn(randomGenerator, 'next').mockImplementation(() => 2)
                      .mockImplementationOnce(() => 1)
                      .mockImplementationOnce(() => 1);
        let game = new Game(randomGenerator);
        let result = game.throwDice(4);
        expect(result).toBe(0);
        expect(spy).toHaveBeenCalled();
        spy.mockRestore();
       }
      )
```

Via jest.spyOn wordt een ‘neppe’ randomGenerator gemaakt
de methode next geef altijd waarde 2 terug
behalve de eerste aanroep (1)
de tweede aanroep (1)

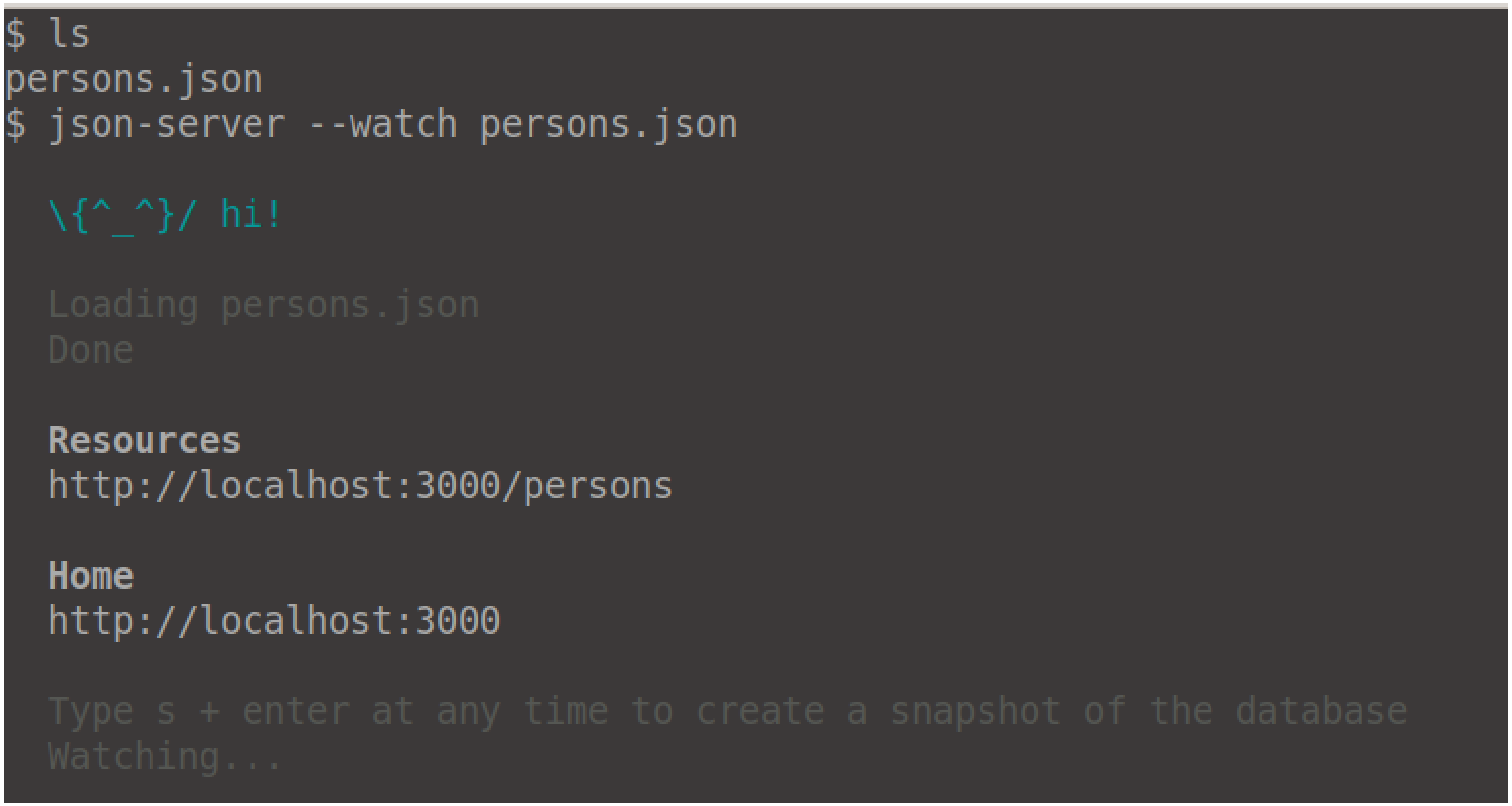
```
$ npm run test
  constructor
    ✓ should generate a Game-object if a valid argument is provided (3 ms)
    ✓ should throw an Error given an invalid argument (2 ms)
    ✓ should throw an Error given an invalid argument (1 ms)
    ✓ should throw an Error given an invalid argument
    ✓ should throw error when 1st parameter is not provided (1 ms)
  throwDice
    ✓ should return 4 if 4 times the same value is thrown (1 ms)
    ✓ should return 0 if two dice throws are not equal (1 ms)

Test Suites: 1 passed, 1 total
Tests:       7 passed, 7 total
Snapshots:   0 total
Time:        1.17 s
Ran all test suites.
```

Voorbeeld3

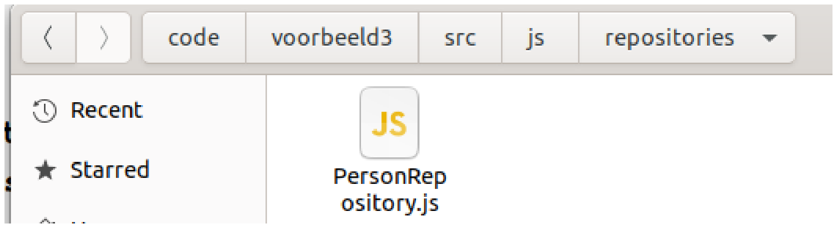
services/PersonService
    methode retrieveNameById

is afhankelijk van
    repositories/PersonRepository

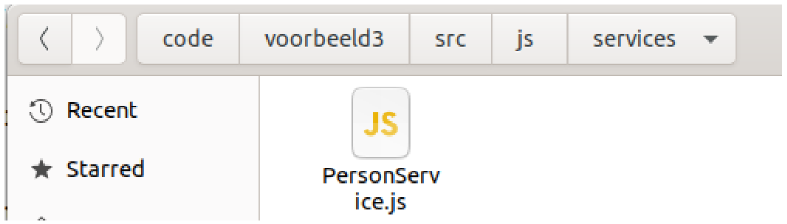
PersonRepository stuurt een request naar json-server



```
$ ls
persons.json
$ json-server --watch persons.json

  \{^_^}/ hi!

  Loading persons.json
  Done

  Resources
  http://localhost:3000/persons

  Home
  http://localhost:3000

  Type s + enter at any time to create a snapshot of the database
  Watching...
```



```
import axios from "axios";

const url = 'http://localhost:3000/persons/';

export default class PersonRepository{
    async findById(id){
        if(!Number.isInteger(id) ||  id<=0){
            throw new Error();
        }
        const response = await axios.get(`${url}${id}`);
        return response.data;
    }
}
```



```
"use strict";

import PersonRepository from '../repositories/PersonRepository';
export default class PersonService{

    #personRepository;

    constructor( personRepository ){
        if (! (personRepository instanceof PersonRepository)){
            throw new Error('Not a PersonRepository');
        }
        this.#personRepository = personRepository;
    }

    async retrieveNameById(id){
        let person = await this.#personRepository.findById(id);
        return person.name.toUpperCase();
    }

}
```

d.i. constructor injection

```
import PersonService from '../../../src/js/services/PersonService';
import PersonRepository from '../../../src/js/repositories/PersonRepository';

const persons = [[{id:1, name:"test1"}],[{id:2, name:"test2"}]];

describe('retrieveNameById',
    () => {
        test.each(persons)(
            'given a person the name in uppercase is returned',
            async (person) => {
                let personRepository = new PersonRepository();
                let spy = jest.spyOn(personRepository, 'findById')
                    .mockImplementation(() => person);
                let personService = new PersonService(personRepository);
                let name = await personService.retrieveNameById(person.id);
                expect(spy).toHaveBeenCalledWith(person.id);
                expect(name).toBe(person.name.toUpperCase());
                spy.mockRestore();
        });

    }
);
```