

# Temmy Alex

Programming Enthusiast

Focus in Web Development

Currently work in KoinWorks as Backend Engineer



**Temmy Alex**

8 years experience as Web Developer

## Education Background



**UNIVERSITAS  
BUDI LUHUR**

**2011-2015**

**Bachelor Degree**

Information System

# Basic Unit Testing

- ☐ Introduction Unit Testing
- ☐ Introduction Jest
- ☐ Install Jest

# Basic Unit Testing



## Introduction Unit Testing



Introduction Jest



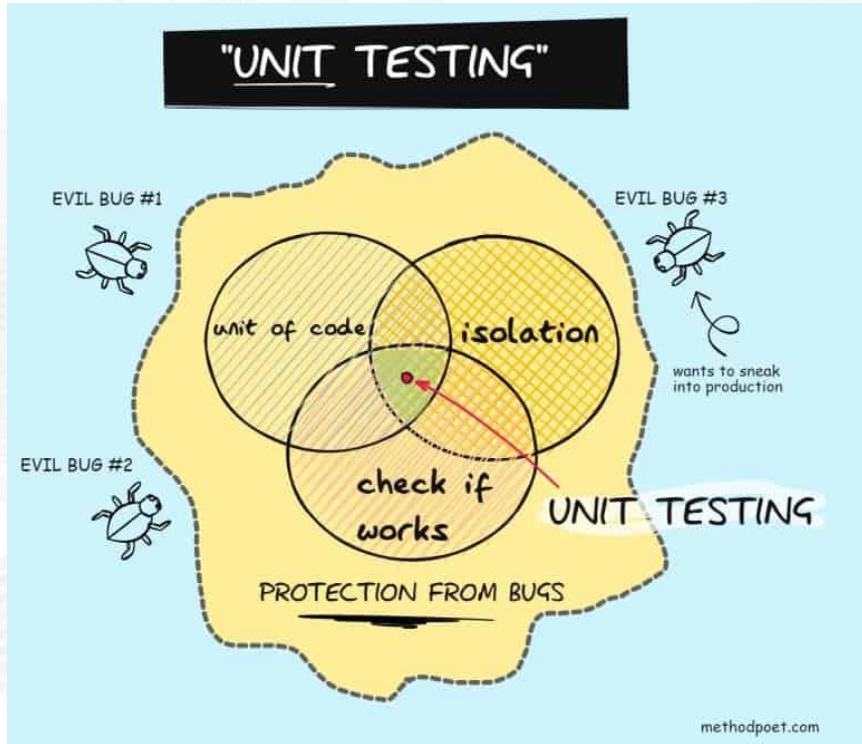
Install Jest

# Apa itu Software Testing?

Software Testing merupakan salah satu ilmu yang terdapat dalam dunia software engineering yang digunakan untuk memastikan kualitas kode dan aplikasi yang dibuat dapat berjalan dengan optimal

Pada pembahasan materi kali ini kita akan membahas bagian dari Ilmu Software Testing yaitu Unit Testing

# Apa itu Unit Testing?



- Unit Testing merupakan bagian dari Software Testing yang berfokus untuk menguji kode bagian terkecil dari program
- Biasanya kode yang dibuat oleh unit testing lebih banyak dari kode fungsi aslinya dikarenakan setiap kondisi-kondisi perlu dilakukan pengecekan unit testing
- Unit Testing berperan penting untuk meningkatkan kualitas kode program (meminimalisir bug)

# Package Unit Testing - NodeJS

Terdapat beberapa package unit testing pada nodejs yang dapat digunakan namun pada pembahasan materi kali ini kita akan menggunakan Jest

Berikut beberapa package unit testing yang dapat digunakan

- Jest (<https://jestjs.io/>)
- Mocha (<https://mochajs.org/>)
- Jasmine (<https://jasmine.github.io/>)
- Testim (<https://www.testim.io/>)



# Basic Unit Testing



Introduction Unit Testing



**Introduction Jest**



Install Jest

# Introduction Jest

- Jest merupakan salah satu library populer yang digunakan untuk Unit Testing pada NodeJs
- Jest merupakan library yang dibuat oleh tim Facebook
- Untuk dokumentasi lengkapnya dapat di cek di <https://jestjs.io/>





# Basic Unit Testing

- ☐ Introduction Unit Testing
- ☐ Introduction Jest



**Install Jest**

# Install Jest

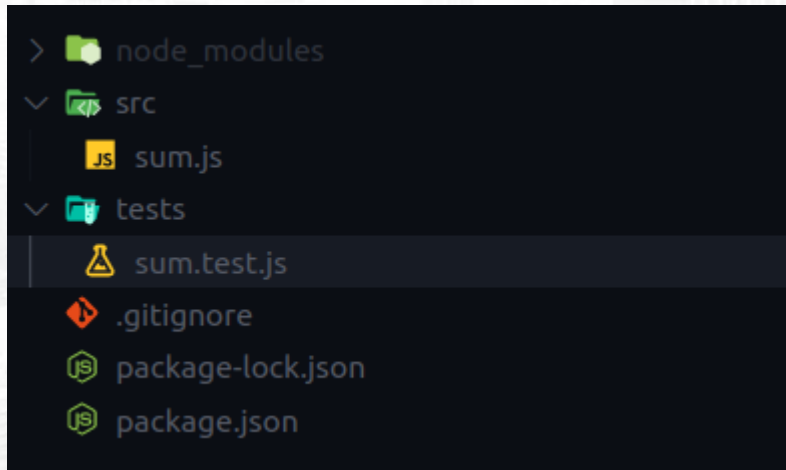
1. Buatlah folder dengan nama unit-testing-example kemudian didalam folder tersebut jalankan perintah npm init

## Activities

```
→ rakamin mkdir unit-testing-example
→ rakamin cd unit-testing-example
→ unit-testing-example npm init
```

# Install Jest

2. Buatlah struktur folder project seperti pada gambar dibawah ini



# Install Jest

3. Kemudian lakukan install package jest dengan menjalankan perintah

`npm install jest`

```
→ unit-testing-example npm install jest  
  
added 277 packages, and audited 278 packages in 12s  
  
30 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities
```

# Install Jest

4. Maka package yang sudah terinstall akan ditambah pada package.json

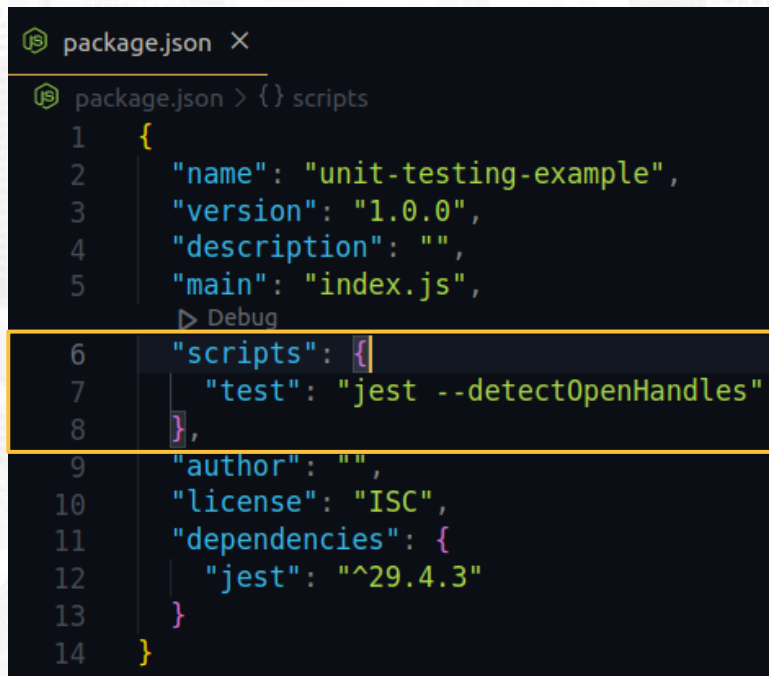
```

package.json X
package.json > ...
1  {
2    "name": "unit-testing-example",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "",
10   "license": "ISC",
11   "dependencies": {
12     "jest": "^29.4.3"
13   }
14 }
15

```

# Install Jest

5. Kemudian lakukan setup pada file `package.json` dan arahkan scripts unit testing menggunakan jest seperti pada gambar dibawah ini (dapat dilihat mulai dari line 6-8)



```

package.json X
package.json > {} scripts
1  {
2    "name": "unit-testing-example",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "jest --detectOpenHandles"
8    },
9    "author": "",
10   "license": "ISC",
11   "dependencies": {
12     "jest": "^29.4.3"
13   }
14 }
  
```



# Study Case

1. Buatlah file dengan nama `sum.js` dan isikan dengan script sum menggunakan javascript
2. Kemudian buatlah file dengan nama `sum.test.js`
3. Kemudian jalankan file `sum.test.js` menggunakan jest

# Study Case Explanation

1. Buatlah file `sum.js` pada folder project yang baru saja kita buat yaitu `unit-testing-example` kemudian isikan file tersebut dengan script dibawah ini

```
JS sum.js X
src > JS sum.js > [e] sumNumber
1   export const sumNumber = (number1, number2) => {
2       return number1 + number2
3   }
```

Script diatas berisi function yang digunakan untuk menjumlahkan angka

# Study Case Explanation

2. Kemudian buatlah file bernama `sum.test.js` dan isikan script dibawah ini

```
JS sum.js X
src > JS sum.js > ...
1  const sumNumber = (number1, number2) => {
2    |    return number1 + number2
3    |
4    |
5    module.exports = sumNumber;
```

# Study Case Explanation

```

sum.test.js ✕
tests > sum.test.js > ...
1  const sumNumber = require('../src/sum');
2
3  test("sumNumber(3, 2) must be 5", () => {
4      const result = sumNumber(3, 2)
5
6      expect(result).toBe(5);
7  })

```

1. Pada line 1 buat const untuk memanggil function yang akan di test
2. Kemudian pada line 3 kita dapat menambahkan deskripsi / keterangan unit test
3. Line 4 digunakan untuk menentukan function yang akan di test
4. Kemudian pada line 6 terdapat function expect untuk memanggil function yang digunakan dan toBe merupakan fungsi pada Jest yang digunakan untuk mengecek hasil sudah sesuai atau belum dengan yang diharapkan

# Study Case Explanation

```

sum.test.js ✕
test > sum.test.js > ...
1  import { sumNumber } from "../src/sum";
2
3  test("sumNumber(1, 2) must be 3", () => {
4    const result = sumNumber(3, 2)
5
6    expect(result).toBe(5);
7  })|

```

1. Pada line 1 digunakan untuk mengimport file yang akan di test
2. Kemudian pada line 3 kita dapat menambahkan deskripsi / keterangan unit test
3. Line 4 digunakan untuk menentukan function yang akan di test
4. Kemudian pada line 6 terdapat function expect untuk memanggil function yang digunakan dan toBe merupakan fungsi pada Jest yang digunakan untuk mengecek hasil sudah sesuai atau belum dengan yang diharapkan

# Study Case Explanation

Kemudian jalankan `npm run test` pada gambar dibawah ini merupakan contoh jika unit test passed / sesuai dengan kriteria yang diharapkan (expected result)

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

● → unit-testing-example npm run test

> unit-testing-example@1.0.0 test
> jest --detectOpenHandles

PASS tests/sum.test.js
  ✓ sumNumber(1, 2) must be 3 (3 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.273 s
Ran all test suites.
```



# Study Case Explanation

Kemudian pada gambar dibawah ini merupakan contoh jika unit test tidak sesuai dengan hasil yang diharapkan

```

④ → unit-testing-example npm run test

> unit-testing-example@1.0.0 test
> jest --detectOpenHandles

FAIL tests/sum.test.js
  ✕ sumNumber(3, 2) must be 5 (6 ms)

  ● sumNumber(3, 2) must be 5

    expect(received).toBe(expected) // Object.is equality

    Expected: 7
    Received: 5

       4 |     const result = sumNumber(3, 2)
       5 |
    >  6 |     expect(result).toBe(7);
         |                       ^
       7 |   })
       8 |
       9 |

      at Object.toBe (tests/sum.test.js:6:20)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
Snapshots:  0 total
Time:        0.218 s, estimated 1 s
Ran all test suites.
  
```

# Study Case Explanation

Untuk penggunaan source code dapat diakses melalui github

<https://github.com/temmy-alex/basic-unit-test-jest>

# Basic Unit Testing



**Introduction Unit Testing**



**Introduction Jest**



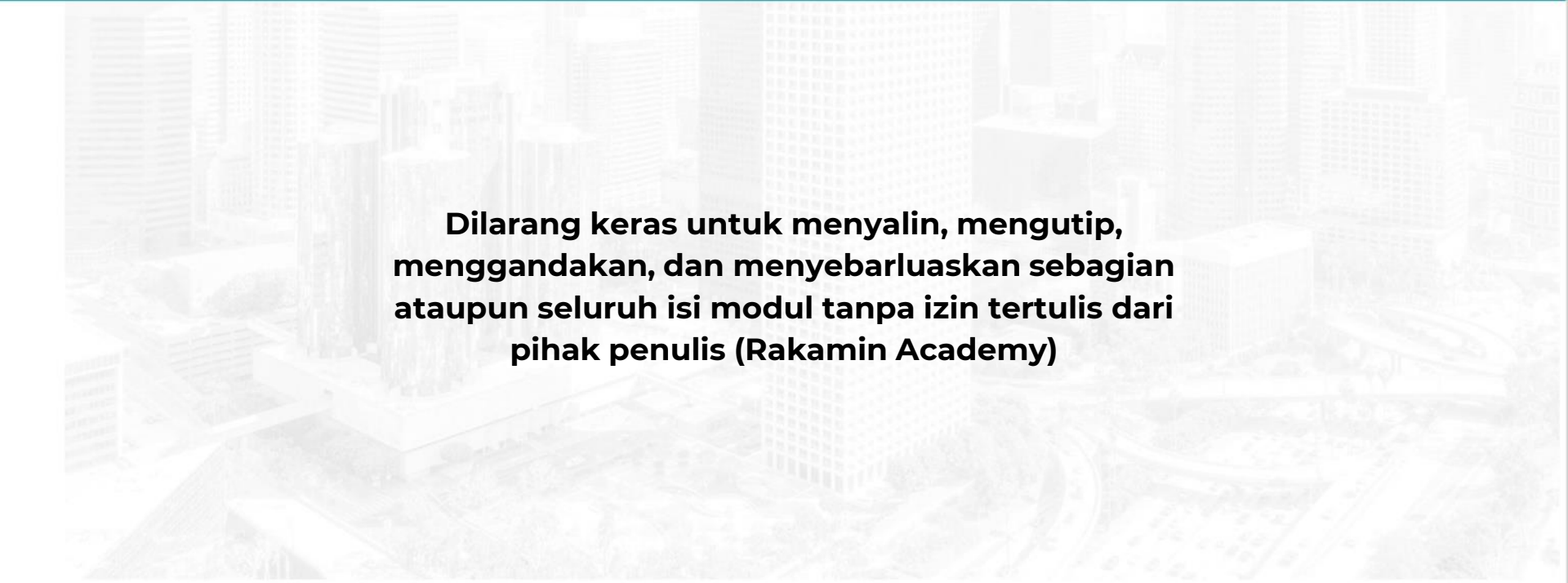
**Install Jest**

# Reference material

1. [Unit Testing Method Poet](#)
2. [Jest Official Documentation](#)
3. [Programmer Zaman Now](#)

**Terima kasih!**

# Copyright Rakamin Academy

A faded, grayscale background image of a city skyline with various skyscrapers and buildings.

**Dilarang keras untuk menyalin, mengutip,  
menggandakan, dan menyebarkan sebagian  
ataupun seluruh isi modul tanpa izin tertulis dari  
pihak penulis (Rakamin Academy)**



# Temmy Alex

Programming Enthusiast

Focus in Web Development

Currently work in KoinWorks as Backend Engineer



**Temmy Alex**

8 years experience as Web Developer

## Education Background



**UNIVERSITAS  
BUDI LUHUR**

**2011-2015**

**Bachelor Degree**

Information System

# Unit Testing with Jest

- ☐ Jest Unit Testing
- ☐ Using Jest in API

# Unit Testing with Jest



**Jest Unit Testing**



Using Jest in API

# Jest - Matchers

Pada pembahasan [slide](#) sebelumnya kita sudah membahas penggunaan function sum dimana didalam function tersebut terdapat 2 parameter yaitu number1 dan number2, dengan kondisi number1 diisi dengan nilai 3 dan number2 diisi dengan nilai 2 dan ekspektasi yang dihasilkan harus memiliki nilai 5 maka kondisi ini dinamakan dengan kondisi matchers

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

# Jest - Matchers

sum.test.js ✕

tests > sum.test.js > ...

```

1  const sumNumber = require('../src/sum');
2
3  test("sumNumber(3, 2) must be 5", () => {
4    const result = sumNumber(3, 2)
5
6    expect(result).toBe(5);
7  })

```

# Jest - Expected

Expected merupakan sebuah function yang digunakan untuk mengembalikan object Matchers dimana object ini digunakan untuk memastikan hasil yang ditampilkan / dikembalikan dari function itu sesuai

Untuk dokumentasi nya dapat di cek

<https://jestjs.io/docs/expect>



# Jest - Expected

Berikut contoh penggunaan Expected menggunakan fungsi `arrayContaining`

array.test.js U X

tests > array.test.js > ...

```
1 describe('programmingFrameworkContaining', () => {
2   const expected = ['Hapi', 'ExpressJS'];
3
4   test('matches even if received contains additional elements', () => {
5     expect(['Hapi', 'ExpressJS', 'KoaJS']).toEqual(expect.arrayContaining(expected));
6   });
7
8   test('does not match if received does not contain expected elements', () => {
9     expect(['Laravel', 'Gin Gonic']).not.toEqual(expect.arrayContaining(expected));
10  });
11  });
```

# Jest - Expected

Pada method `arrayContaining` digunakan untuk memastikan apakah value yang dihasilkan sesuai dengan list value array yang terdapat pada variable `expected` atau tidak


# Jest - Equals Matchers



Equals Matchers merupakan fungsi yang digunakan untuk memastikan sama dengan ekspektasi yang kita set baik pada function maupun variable

Terdapat 2 fungsi yang biasanya digunakan untuk Equals Matchers

Fungsi	Keterangan
<code>expect(value).toBe(expected)</code>	Value sama dengan expected yang biasanya digunakan untuk value (bukan object)
<code>expect(value).toEqual(expected)</code>	Value sama dengan expected yang dimana fungsi ini digunakan untuk membandingkan semua properties

# Jest - Equals Matchers

 equals.test.js U X

tests >  equals.test.js >  test("test using toEquals") callback

```

1  test("test using toBe", () => {
2    let name = "NodeJS";
3    let greeting = `Hello ${name}`;
4
5    expect(greeting).toBe("Hello NodeJS")
6  })
7
8  test("test using toEquals", () => {
9    let frameworkJS = {id: "N1"};
10   Object.assign(frameworkJS, {name: "NodeJS"})
11
12   expect(frameworkJS).toEqual({id: "N1", name: "NodeJS"})
13 })

```

# Jest - Truthiness Matchers

Truthiness merupakan salah satu Matchers yang digunakan untuk melakukan pengecekan nilai undefined, null dan false berikut beberapa function yang dapat digunakan

Fungsi	Keterangan
<code>expect(value).toBeNull()</code>	Memastikan value adalah null
<code>expect(value).toBeUndefined()</code>	Memastikan value adalah undefined

# Jest - Truthiness Matchers

Fungsi	Keterangan
<code>expect(value).toBeNull()</code>	Memastikan value adalah null
<code>expect(value).toBeUndefined()</code>	Memastikan value adalah undefined
<code>expect(value).toBeDefined()</code>	Kebalikan dari fungsi <code>toBeUndefined</code>
<code>expect(value).toBeTruthy()</code>	Memastikan value apapun bernilai true, jika if statement menganggap true
<code>expect(value).toBeFalsy()</code>	Memastikan value apapun bernilai false, jika if statement menganggap false



# Jest - Truthiness Matchers

```

truthiness.test.js U X
tests > truthiness.test.js > test("using truthiness") callback
1  test("using truthiness", () => {
2      let value = null;
3      expect(value).toBeNull()
4      expect(value).toBeDefined()
5      expect(value).toBeFalsy()
6
7      value = undefined;
8      expect(value).toBeUndefined()
9      expect(value).toBeFalsy()
10
11     value = "NodeJS"
12     expect(value).toBeTruthy()
13 })
  
```



# Jest - Truthiness Matchers

```

truthiness.test.js U X
tests > truthiness.test.js > test("using truthiness") callback
1  test("using truthiness", () => {
2      let value = null;
3      expect(value).toBeNull()
4      expect(value).toBeDefined()
5      expect(value).toBeFalsy()
6
7      value = undefined;
8      expect(value).toBeUndefined()
9      expect(value).toBeFalsy()
10
11     value = "NodeJS"
12     expect(value).toBeTruthy()
13 })
  
```

# Jest - Number Matchers

Terdapat Matchers pada Jest yang dapat digunakan untuk melakukan pengecekan pada value berupa number dan tentunya untuk pengecekan value number ini dapat juga menggunakan fungsi `toBe()` dan `toEqual()` yang digunakan untuk memastikan value sama dengan expected

# Jest - Number Matchers

Fungsi	Keterangan
<code>toBeGreaterThan(n)</code>	Memastikan value lebih besar dari n
<code>toBeGreaterThanOrEqual(n)</code>	Memastikan value lebih besar / sama dengan n
<code>toBeLessThan(n)</code>	Memastikan value lebih kecil dari n
<code>toBeLessThanOrEqual(n)</code>	Memastikan value lebih kecil / sama dengan n

# Jest - Number Matchers

number.test.js U X

tests > number.test.js > ...

```

1  test("numbers", () => {
2    const value = 2 + 2
3    expect(value).toBeGreaterThan(3)
4    expect(value).toBeGreaterThanOrEqual(3.5)
5    expect(value).toBeLessThan(5)
6    expect(value).toBeLessThanOrEqual(4.5)
7
8    expect(value).toBe(4)
9    expect(value).toEqual(4)
10 })

```

# Jest - String Matchers

Terdapat Matchers pada Jest yang dapat digunakan untuk melakukan pengecekan pada string berupa number dan tentunya untuk pengecekan value string ini dapat juga menggunakan fungsi `toBe()` dan `toEqual()` yang digunakan untuk memastikan value sama dengan expected

# Jest - Number Matchers

Jest juga memiliki matchers function yang digunakan untuk melakukan pengecekan value dalam sebuah array, kita juga dapat menggunakan fungsi yang sama pada numbers dan string yaitu `toEqual()`



# Jest - String Matchers

Fungsi	Keterangan
<code>toMatch(regex)</code>	Memastikan value sesuai dengan regex



# Jest - String Matchers

string.test.js U X

tests > string.test.js > test("string") callback

```

1  test("string", () => {
2    const frameworkJS = "NodeJS"
3
4    expect(frameworkJS).toBe("NodeJS")
5    expect(frameworkJS).toEqual("NodeJS")
6    expect(frameworkJS).toMatch(/ode/)
7  })

```

# Jest - Array Matchers

Jest juga memiliki matchers function yang digunakan untuk melakukan pengecekan value dalam sebuah array, kita juga dapat menggunakan fungsi yang sama pada numbers dan string yaitu `toEqual()`

# Jest - Array Matchers

Fungsi	Keterangan
<code>toContain(item)</code>	Memastikan value array memiliki item dimana pengecekan item menggunakan item <code>toBe()</code>
<code>toContainEqual(item)</code>	Memastikan value array memiliki item dimana pengecekan item menggunakan item <code>toEqual()</code>

# Jest - Array Matchers

array.test.js U X

tests > array.test.js > ...

```

1  test("array", () => {
2      const frameworkJS = ["ExpressJS", "Hapi", "Koa"]
3      expect(frameworkJS).toContain("ExpressJS")
4      expect(frameworkJS).toEqual(["ExpressJS", "Hapi", "Koa"])
5
6      const unitTestingLibrary = [{title: "Jest"}, {title: "Mocha"}]
7      expect(unitTestingLibrary).toContainEqual({title: "Jest"})
8      expect(unitTestingLibrary).toEqual([{title: "Jest"}, {title: "Mocha"}])
9  })

```

# Unit Testing with Jest



Jest Unit Testing



**Using Jest in API**

# Using Jest in API

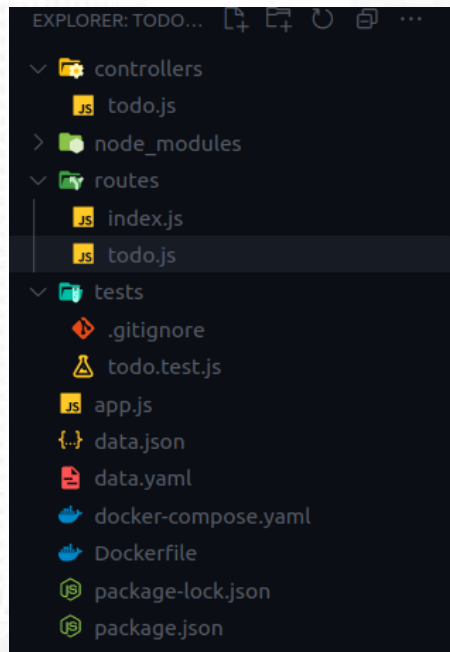
Jest juga dapat digunakan untuk project pembuatan API, dimana pembahasan kali ini kita akan menggunakan expressJS dengan Jest untuk melakukan pengecekan unit testing

Untuk contoh API yang akan digunakan dapat di clone melalui link dibawah ini

<https://github.com/temmy-alex/unit-test-todoapi>

# Using Jest in API

1. Tambahkan folder tests serta buatlah file bernama `todo.test.js` dan pastikan struktur direktori project seperti dibawah ini





# Using Jest in API

2. Kemudian isikan file todo.test.js dengan script dibawah ini

```

todo.test.js ×
tests > todo.test.js > ...
1  const app = require('../app')
2  const request = require('supertest');
3
4  test('Get 1 Data From List', (done) => {
5    request(app)
6      .get('/api/v1/todo')
7      .expect('Content-Type', /json/)
8      .expect(200)
9      .then(response => {
10        expect(response.body.message).toBe("Success")
11        done()
12      })
13    .catch(done)
14  })

```

# Using Jest in API

```

todo.test.js x
tests > todo.test.js > ...
1  const app = require('../app')
2  const request = require('supertest');
3
4  test('Get 1 Data From List', (done) => {
5      request(app)
6          .get('/api/v1/todo')
7          .expect('Content-Type', /json/)
8          .expect(200)
9          .then(response => {
10             expect(response.body.message).toBe("Success")
11             done()
12         })
13         .catch(done)
14 })
  
```

3. Pada line 1 terdapat pemanggilan app yang digunakan untuk memanggil file app.js yang digunakan untuk menjalankan aplikasi

Serta pada line 2 terdapat variable request dimana untuk case api ini kita menggunakan library supertest

Kemudian untuk expected value yang di cek adalah tipe data harus berbentuk json seperti yang tertera pada line 7

Serta api harus memiliki status 200 yang artinya api berhasil ditampilkan yang ada di line 8

Kemudian pada bagian line 9 - 10 berfokus untuk mengecek isi dari data response yang dihasilkan api dimana jika value dari response message success maka expected value nya sudah terpenuhi

# Using Jest in API

4. Dan jalankan npm run test untuk menjalankan unit test

```

• → todoapi git:(master) npm run test

> todoapi@1.0.0 test
> jest --detectOpenHandles

PASS tests/todo.test.js
  ✓ Get 1 Data From List (76 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.319 s, estimated 1 s
Ran all test suites.
  
```

# Study Case

Tambahkan unit testing yang digunakan untuk memastikan jumlah data pada halaman pertama serta mengambil 1 sample data dari api dan memastikan value yang dihasilkan sesuai expected

# Study Case Explanation

1. Menambahkan unit testing untuk jumlah data per halaman

Tambahkan script pada file `todo.test.js` seperti gambar dibawah ini

```
test('Get Count Data From List', (done) => {
  request(app)
    .get('/api/v1/todo')
    .expect('Content-Type', /json/)
    .expect(200)
    .then(response => {
      expect(response.body.data.length).toBe(5)
      done()
    })
    .catch(done)
})
```

# Study Case Explanation

```
test('Get Count Data From List', (done) => {
  request(app)
    .get('/api/v1/todo')
    .expect('Content-Type', /json/)
    .expect(200)
    .then(response => {
      expect(response.body.data.length).toBe(5)
      done()
    })
    .catch(done)
})
```

Pada bagian script berikut untuk melakukan pengecekan jumlah data pada response api, kita dapat menggunakan fungsi expect dan fungsi expect tersebut dapat kita isi dengan `response.body.data.length` yang digunakan untuk mengambil jumlah data yang dihasilkan pada response api todo

Pada halaman pertama api todo memiliki jumlah response data todo sebanyak 5 maka value yang digunakan sesuai dengan ekspektasi (pada fungsi expect)

# Study Case Explanation

2. Menambahkan unit testing untuk mengambil 1 data

Tambahkan script pada file `todo.test.js` seperti gambar dibawah ini

```
test('Get 1 Data From List', (done) => {
  request(app)
    .get('/api/v1/todo')
    .expect('Content-Type', /json/)
    .expect(200)
    .then(response => {
      expect(response.body.data[0].title).toBe("Todo 1")
      done()
    })
    .catch(done)
})
```



# Study Case Explanation

```
test('Get 1 Data From List', (done) => {
  request(app)
    .get('/api/v1/todo')
    .expect('Content-Type', /json/)
    .expect(200)
    .then(response => {
      expect(response.body.data[0].title).toBe("Todo 1")
      done()
    })
    .catch(done)
})
```

Pada bagian script berikut terdapat unit testing yang digunakan untuk mengambil salah satu data dari response api dimana value yang akan digunakan untuk pengecekan adalah title, maka untuk mengambil salah satu value yang terdapat pada api kita dapat menggunakan index pada response key data pada api yaitu data pertama memiliki index ke 0

# Study Case Explanation

- Untuk memastikan unit testing yang ditambahkan berhasil jalankan `npm run test`

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

● → todoapi git:(master) x npm run test

> todoapi@1.0.0 test
> jest --detectOpenHandles

PASS tests/todo.test.js
  ✓ Get Success Status From List (75 ms)
  ✓ Get Count Data From List (9 ms)
  ✓ Get 1 Data From List (6 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.393 s, estimated 1 s
Ran all test suites.

```

# Unit Testing with Jest



**Jest Unit Testing**



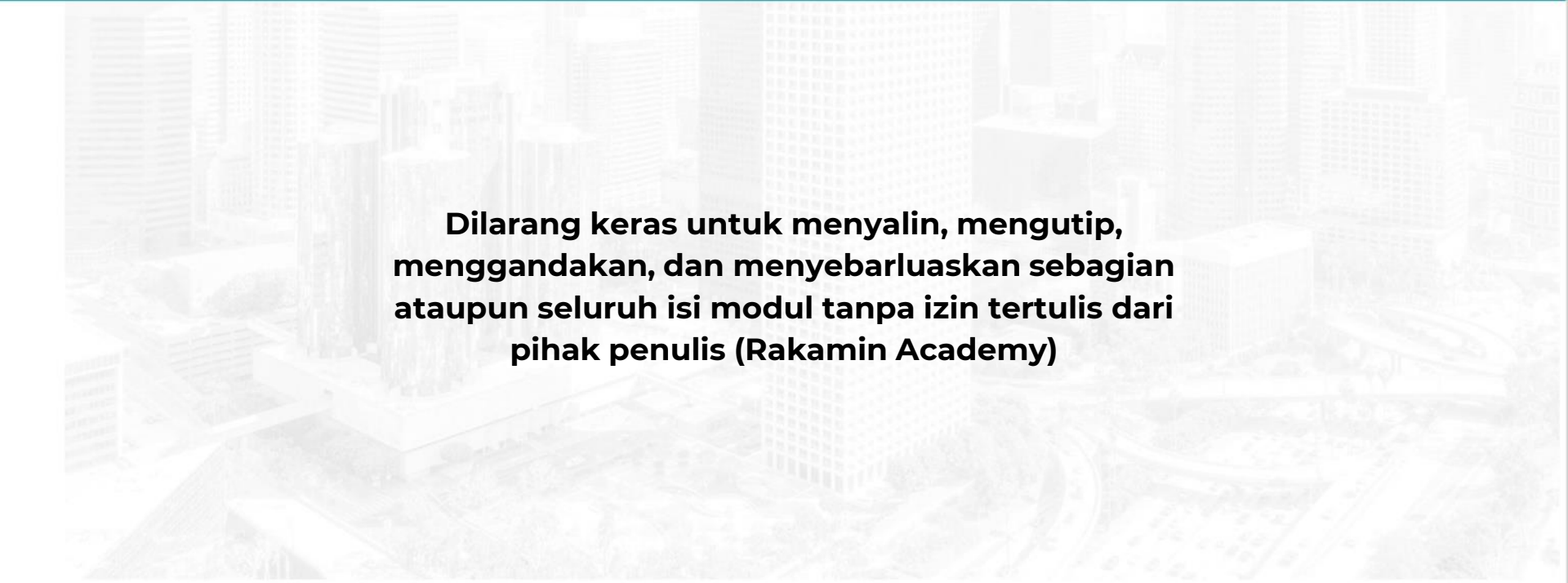
**Using Jest in API**

# Reference material

1. [Unit Testing Method Poet](#)
2. [Jest Official Documentation](#)
3. [Programmer Zaman Now](#)

# **Terima kasih!**

# Copyright Rakamin Academy

A faded, light-colored background image of a city skyline with various skyscrapers and buildings.

**Dilarang keras untuk menyalin, mengutip,  
menggandakan, dan menyebarkan sebagian  
ataupun seluruh isi modul tanpa izin tertulis dari  
pihak penulis (Rakamin Academy)**