
NodeJS Unit Test

Eko Kurniawan Khannedy

Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 11+ years experiences
- www.programmerzamannow.com
- youtube.com/c/ProgrammerZamanNow



Eko Kurniawan Khannedy

- Telegram : [@khannedy](https://t.me/khannedy)
- Facebook : [fb.com/ProgrammerZamanNow](https://www.facebook.com/ProgrammerZamanNow)
- Instagram : [instagram.com/programmerzamannow](https://www.instagram.com/programmerzamannow)
- Youtube : [youtube.com/c/ProgrammerZamanNow](https://www.youtube.com/c/ProgrammerZamanNow)
- Telegram Channel : t.me/ProgrammerZamanNow
- Email : echo.khannedy@gmail.com

Sebelum Belajar

- Mengikuti Kelas JavaScript Programmer Zaman Now
- NodeJS Dasar
- NodeJS Package Manager



Agenda

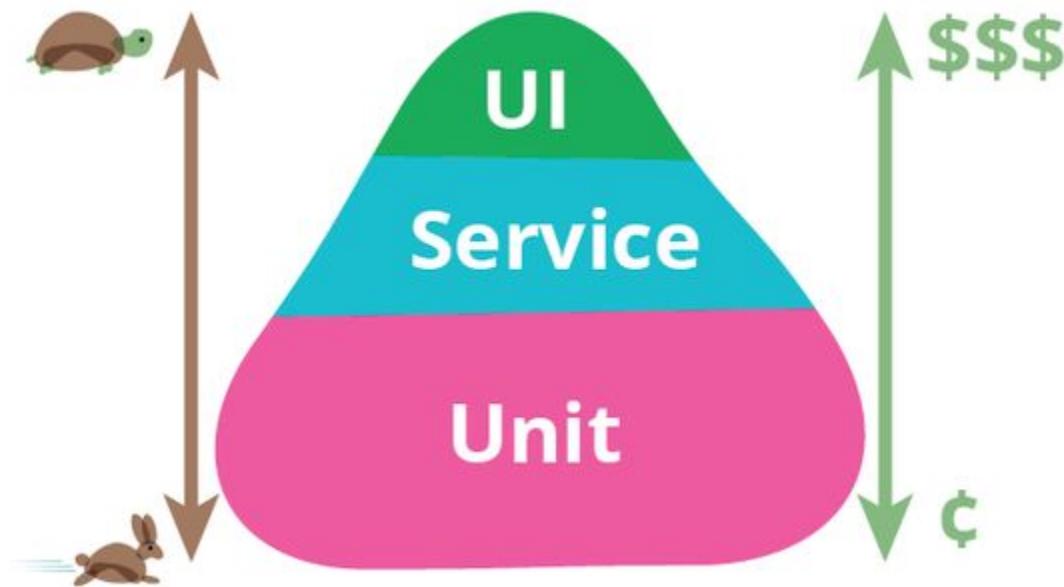
- Pengenalan Testing
- Berkenalan dengan Jest Library
- Membuat Unit Test
- Matchers
- Mock
- Dan lain-lain

Pengenalan Software Testing

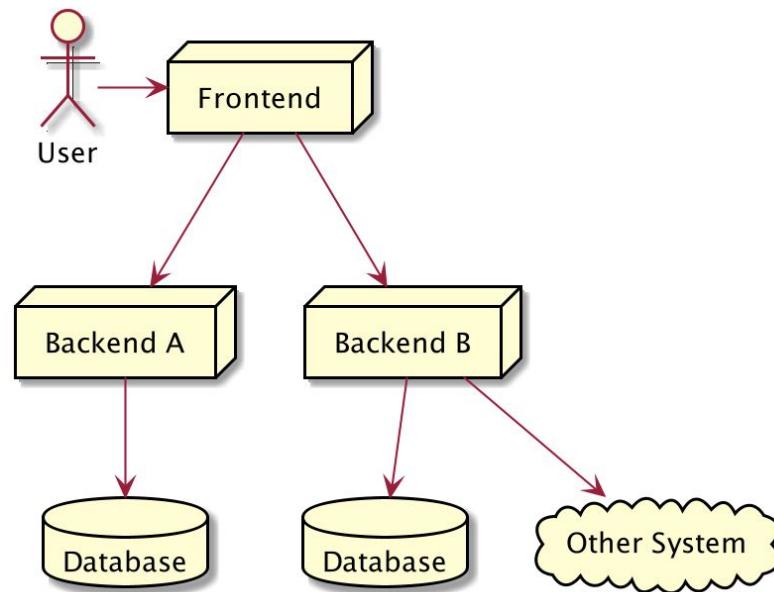
Pengenalan Software Testing

- Software testing adalah salah satu disiplin ilmu dalam software engineering
- Tujuan utama dari software testing adalah memastikan kualitas kode dan aplikasi kita baik
- Ilmu untuk software testing sendiri sangatlah luas, pada materi ini kita hanya akan fokus ke unit testing

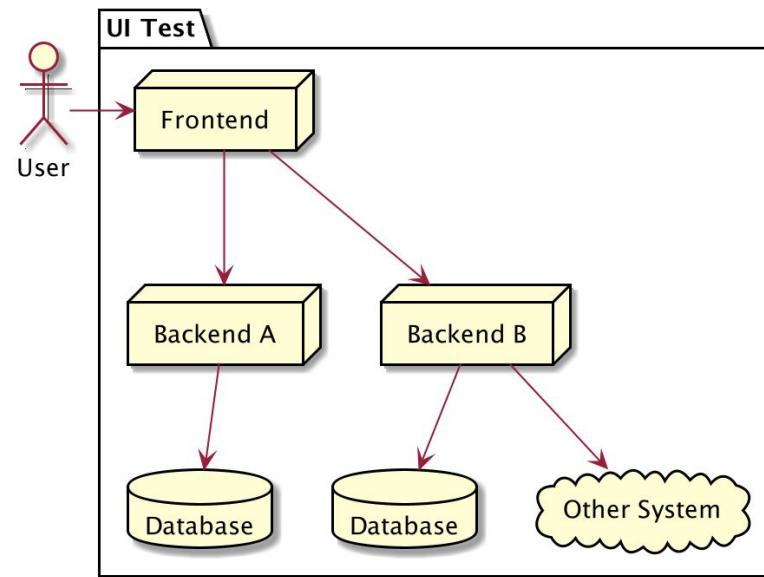
Test Pyramid



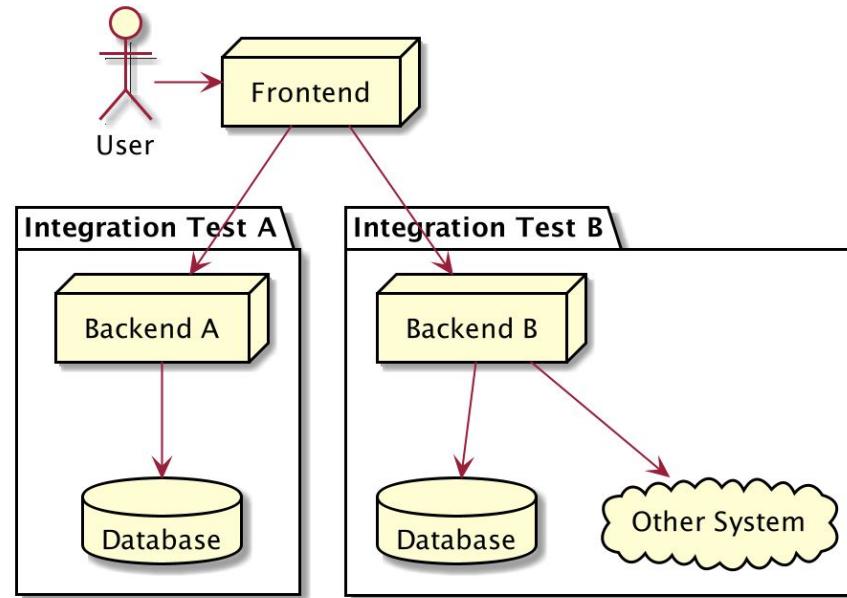
Contoh High Level Architecture Aplikasi



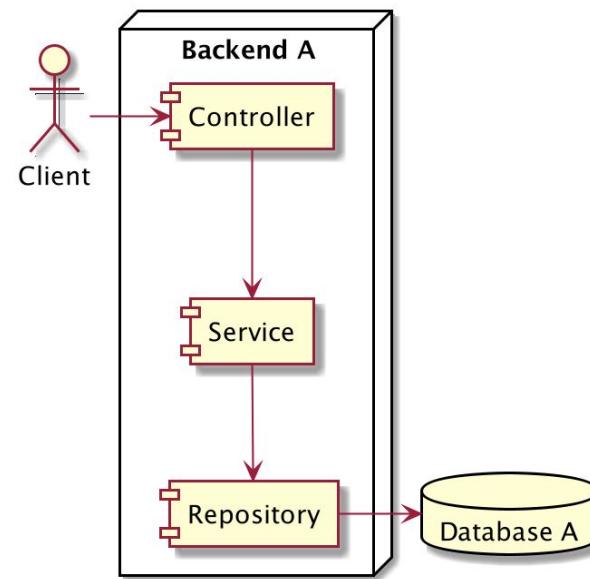
UI Test / End to End Test



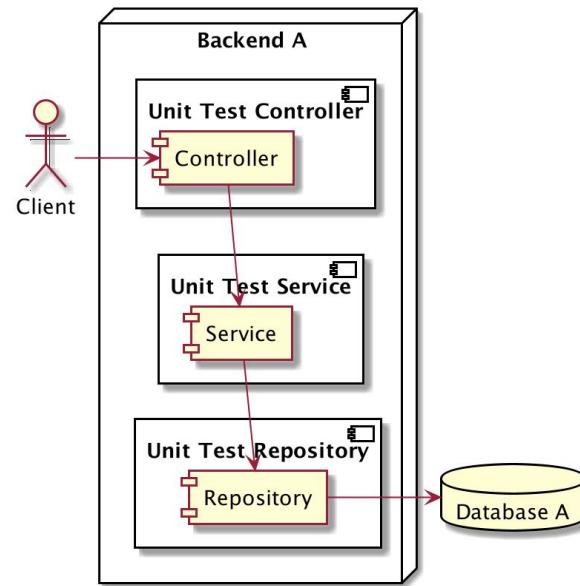
Service Test / Integration Test



Contoh Internal Architecture Aplikasi



Unit Test



Unit Test

- Unit test akan fokus menguji bagian kode program terkecil, biasanya menguji sebuah method
- Unit test biasanya dibuat kecil dan cepat, oleh karena itu biasanya kadang kode unit test lebih banyak dari kode program aslinya, karena semua skenario pengujian akan dicoba di unit test
- Unit test bisa digunakan sebagai cara untuk meningkatkan kualitas kode program kita

Pengenalan Jest

NodeJS Assertion Package

- NodeJS sendiri sebenarnya memiliki package untuk melakukan assertion, namun kita tidak akan membahasnya, karena jarang sekali orang menggunakan package tersebut
- <https://nodejs.org/api/assert.html>
- Programmer NodeJS kebanyakan menggunakan library yang lebih baik untuk melakukan unit test

NodeJS Unit Test Library

Sebenarnya ada banyak sekali library opensource yang bisa kita gunakan untuk melakukan unit test di NodeJS

- Jest : <https://jestjs.io/>
- Mocha : <https://mochajs.org/>
- Jasmine : <https://jasmine.github.io/>
- Dan masih banyak yang lainnya

Pengenalan Jest

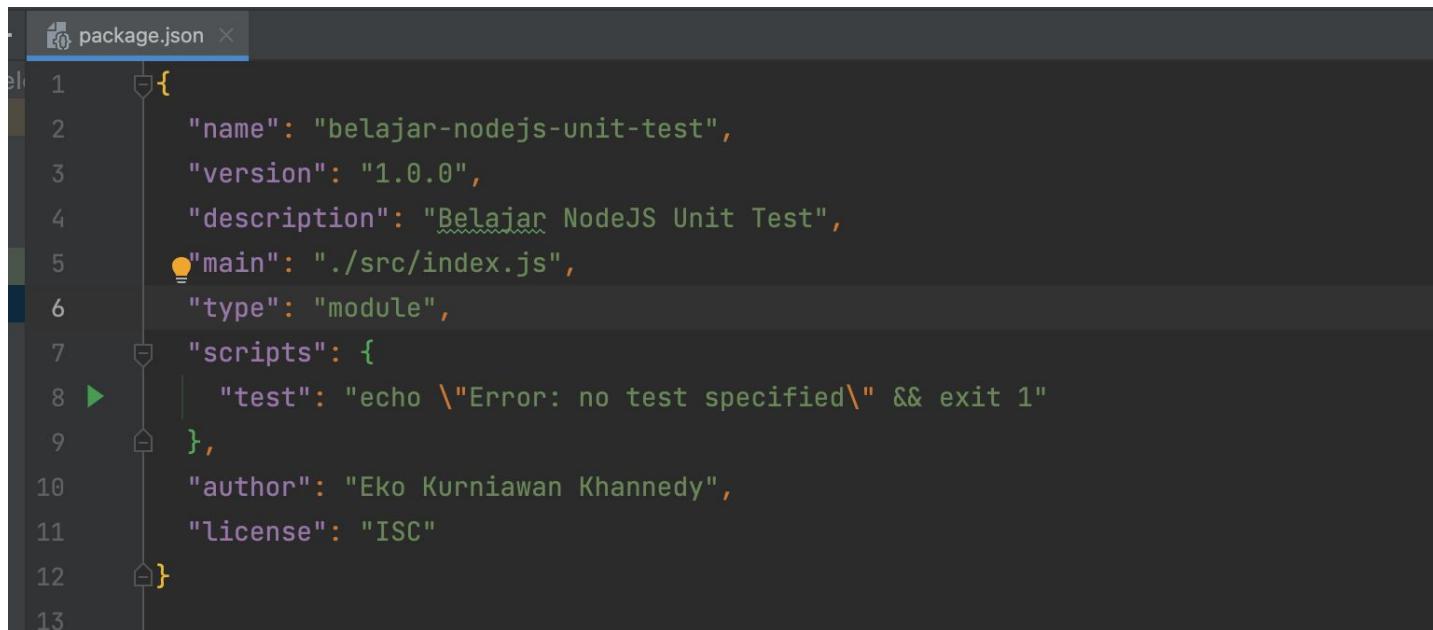
- Jest adalah salah satu library untuk unit test NodeJS yang sangat populer
- Jest sendiri dibuat oleh Facebook
- Jest terintegrasi sangat baik dengan banyak teknologi seperti NodeJS, ReactJS, VueJS, dan lain-lain
- Jest fokus pada kesederhanaan, sehingga penggunaannya sangat mudah untuk pemula yang ingin mencoba unit test
- <https://jestjs.io/>

Membuat Project

Membuat Project

- npm init

Kode : package.json



A screenshot of a code editor showing a package.json file. The file contains the following JSON code:

```
1  {  
2      "name": "belajar-nodejs-unit-test",  
3      "version": "1.0.0",  
4      "description": "Belajar NodeJS Unit Test",  
5      "main": "./src/index.js",  
6      "type": "module",  
7      "scripts": {  
8          "test": "echo \\\"Error: no test specified\\\" && exit 1"  
9      },  
10     "author": "Eko Kurniawan Khannedy",  
11     "license": "ISC"  
12 }  
13
```

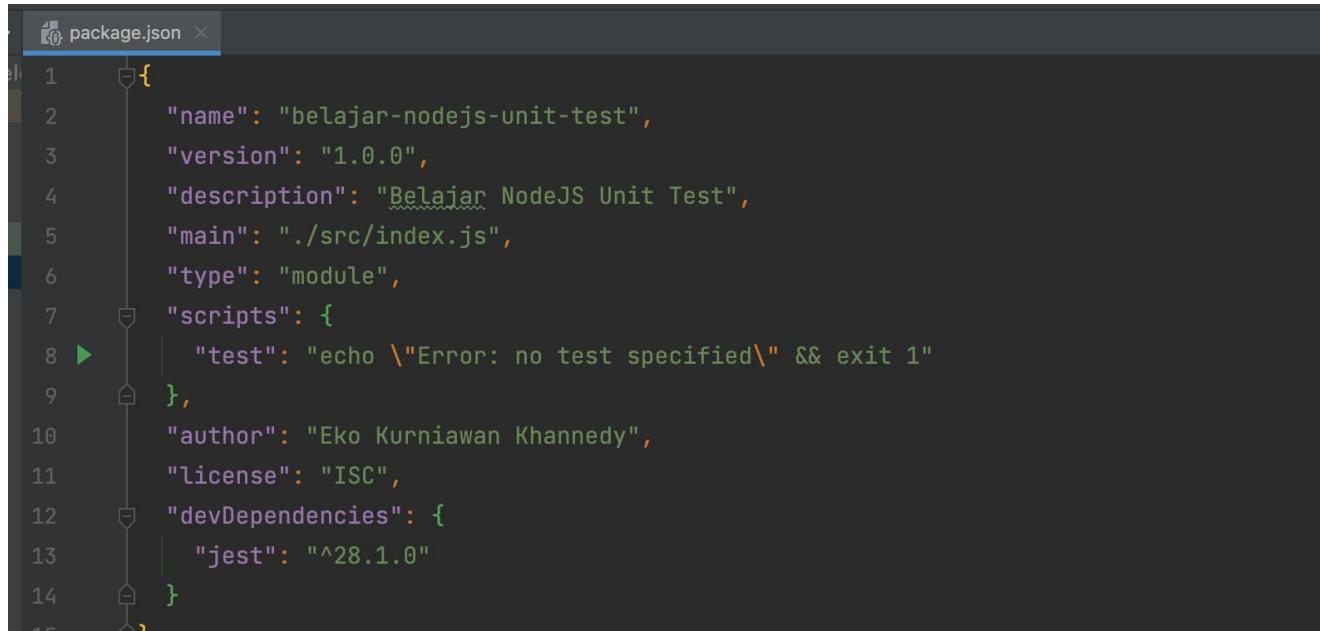
The code editor has syntax highlighting and shows line numbers from 1 to 13. A yellow lightbulb icon is placed next to the "main" field, indicating a potential issue or suggestion.

Menginstall Jest

Menginstall Jest

- Jest digunakan untuk membuat unit test saja, sehingga kita tidak perlu menambahkan sebagai dependency production
- Kita cukup tambahkan sebagai development dependency
- Kita bisa tambahkan di package.json atau gunakan perintah :
npm install jest --save-dev
- <https://www.npmjs.com/package/jest>

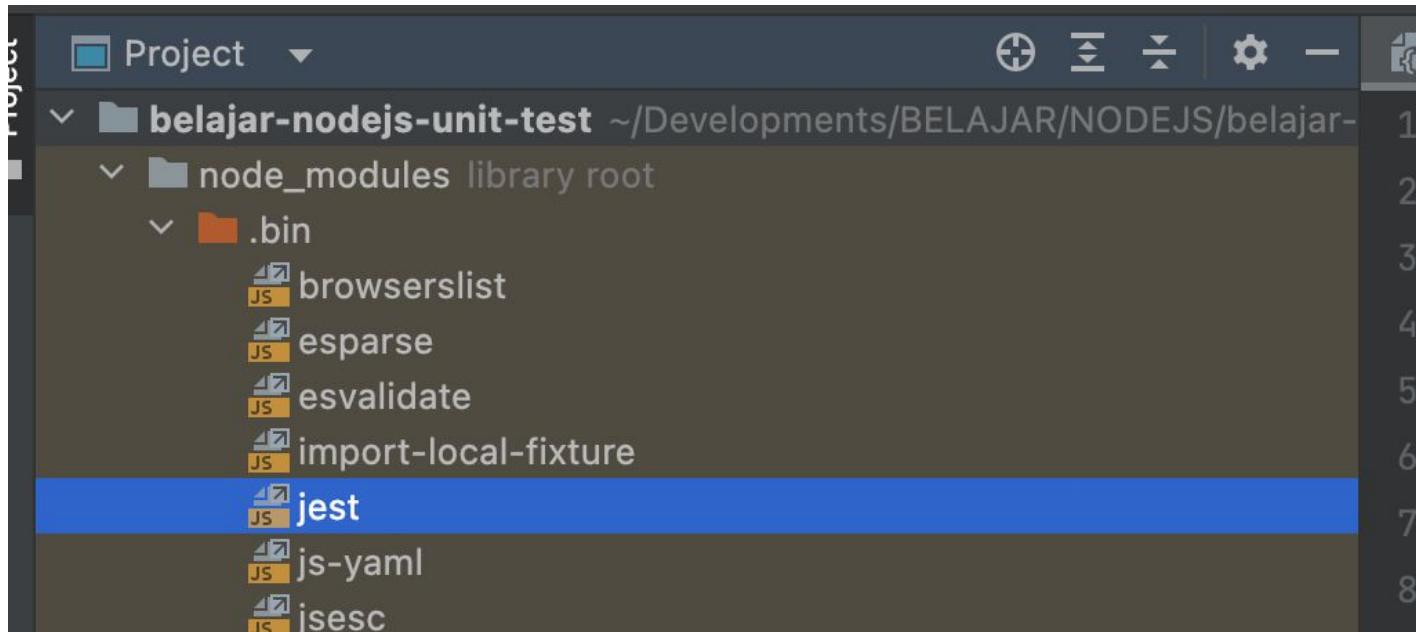
Kode : package.json



```
package.json ×

1  {
2      "name": "belajar-nodejs-unit-test",
3      "version": "1.0.0",
4      "description": "Belajar NodeJS Unit Test",
5      "main": "./src/index.js",
6      "type": "module",
7      "scripts": {
8          "test": "echo \\\"Error: no test specified\\\" && exit 1"
9      },
10     "author": "Eko Kurniawan Khannedy",
11     "license": "ISC",
12     "devDependencies": {
13         "jest": "^28.1.0"
14     }
15 }
```

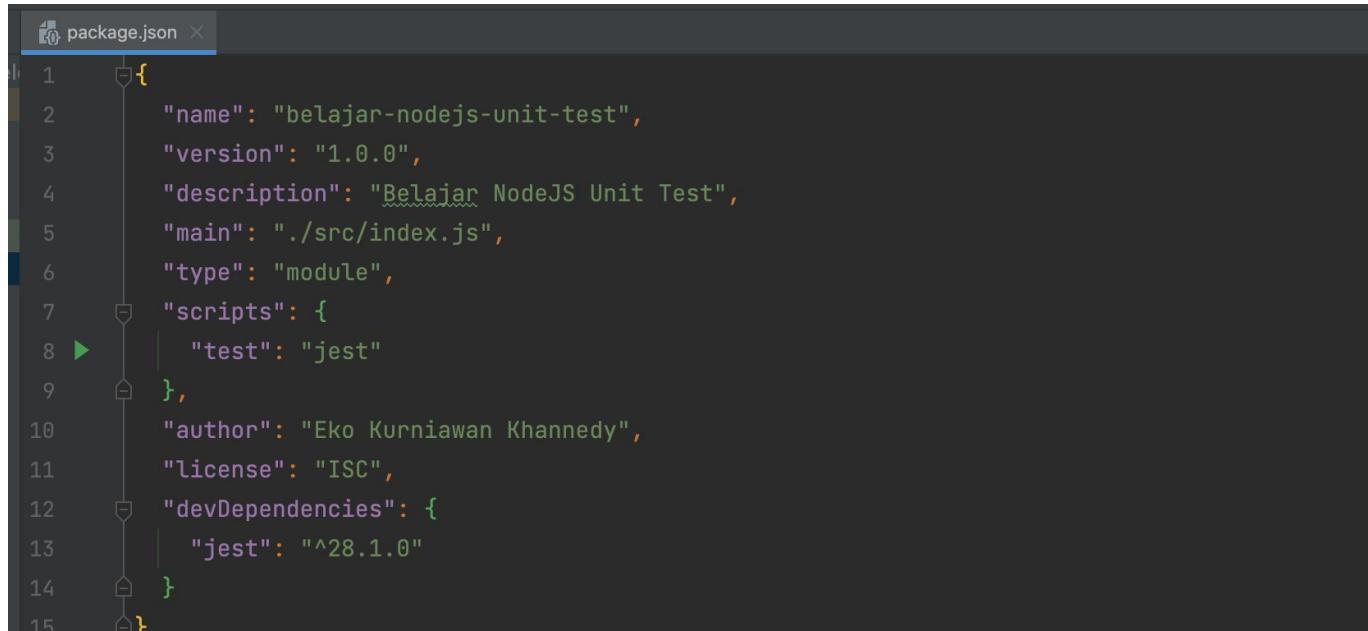
Program Jest



Menjalankan Unit Test

- Untuk menjalankan unit test menggunakan Jest, kita bisa jalankan file jest yang ada di node_modules/.bin/jest
- Tapi kita bisa permudah menggunakan script di package, cukup ketikkan kode program nya saja di bagian script test

Kode : package.json



```
package.json x
1  {
2      "name": "belajar-nodejs-unit-test",
3      "version": "1.0.0",
4      "description": "Belajar NodeJS Unit Test",
5      "main": "./src/index.js",
6      "type": "module",
7      "scripts": {
8          "test": "jest"
9      },
10     "author": "Eko Kurniawan Khannedy",
11     "license": "ISC",
12     "devDependencies": {
13         "jest": "^28.1.0"
14     }
15 }
```

Kode : Menjalankan Unit Test

```
Terminal: Local × + ▾
→ belajar-nodejs-unit-test npm test

> belajar-nodejs-unit-test@1.0.0 test
> jest

No tests found, exiting with code 1
Run with `--passWithNoTests` to exit with code 0
In /Users/khannedy/Developments/BELAJAR/NODEJS/belajar-nodejs-unit-test
  2 files checked.
  testMatch: **/_tests_/**/*.[jt]s?(x), **/?(*.)+(spec|test).[tj]s?(x) - 0 matches
  testPathIgnorePatterns: /node_modules/ - 2 matches
  testRegex: - 0 matches
  Pattern: - 0 matches

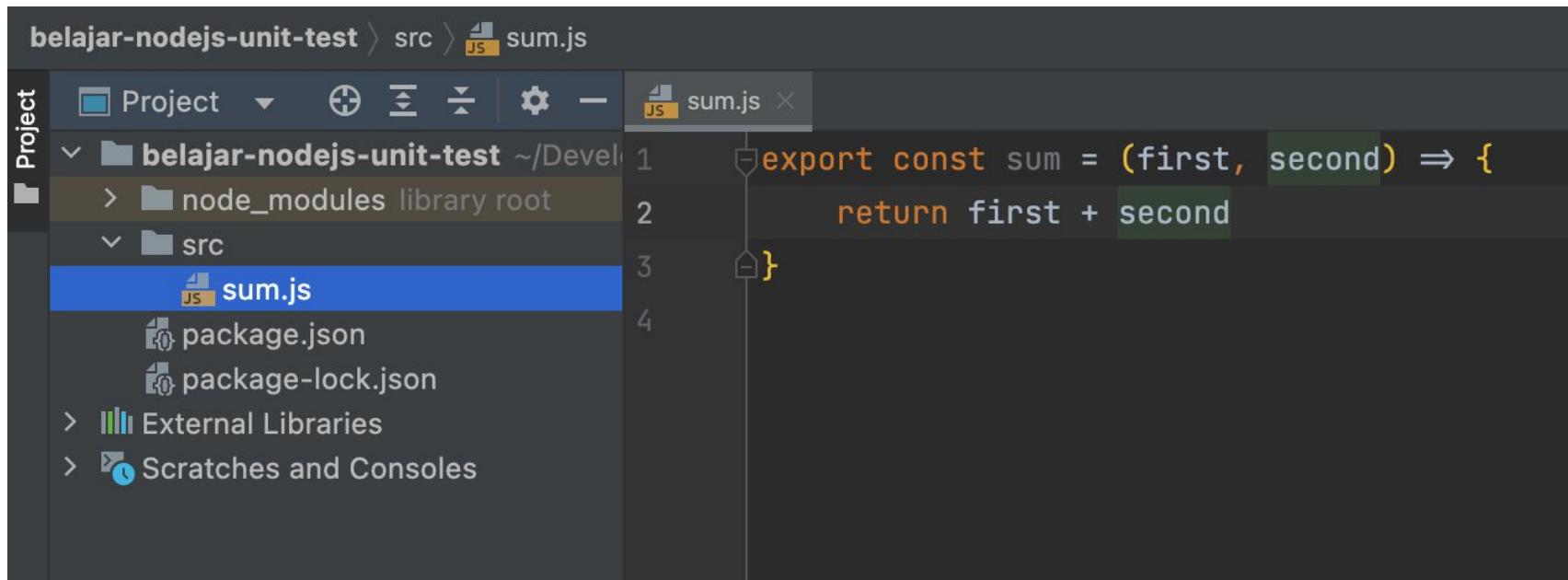
→ belajar-nodejs-unit-test
```

NPX

- Menjalankan perintah jest lumayan ribet karena kita harus selalu menjalankan melalui folder node_modules/.bin/
- Untungnya, di NodeJS terdapat program bernama NPX (Node Package Runner)
- NPX ini digunakan spesial untuk menjalankan perintah yang bisa secara otomatis mendetect file yang terdapat di node_modules/.bin/
- Jadi untuk menjalankan Jest, kita bisa menggunakan perintah :
`npx jest`

Membuat Unit Test

Kode : Sum Function



The screenshot shows a code editor interface with the following details:

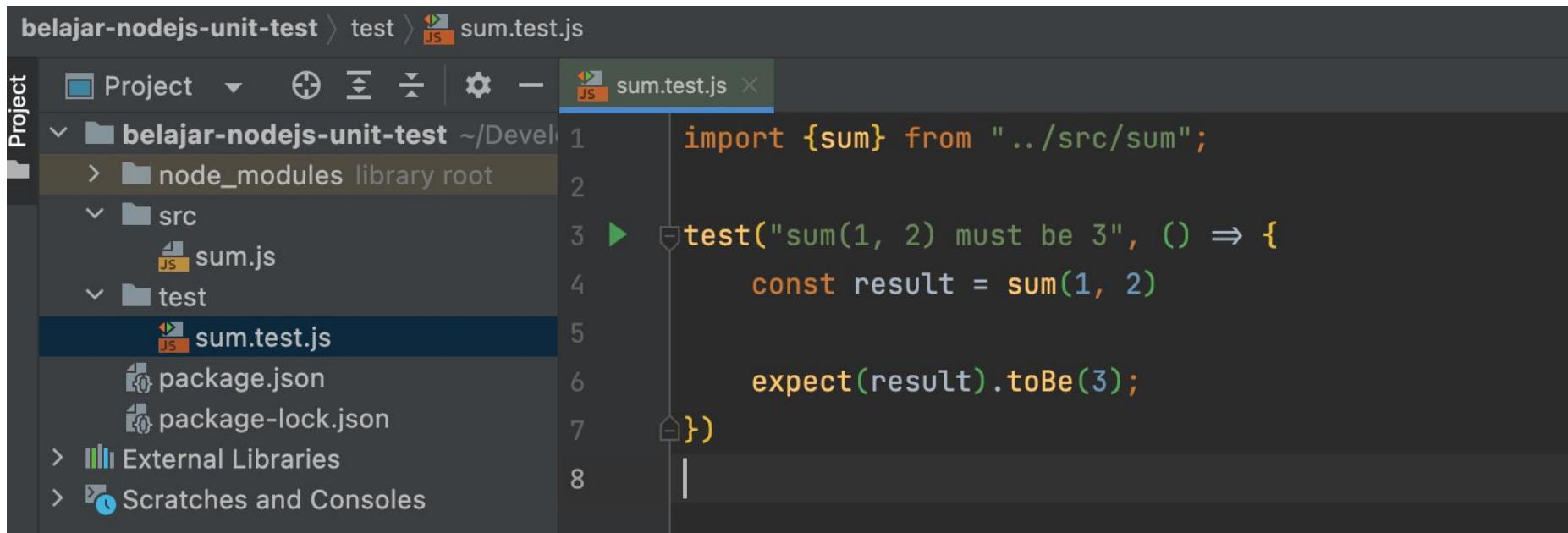
- Project Bar:** Shows the project path: `belajar-nodejs-unit-test > src > sum.js`.
- Project Explorer:** Displays the project structure:
 - `belajar-nodejs-unit-test` (root)
 - `node_modules` (library root)
 - `src`
 - `sum.js` (selected file, highlighted in blue)
 - `package.json`
 - `package-lock.json`
 - `External Libraries`
 - `Scratches and Consoles`
- Code Editor:** The file `sum.js` contains the following code:

```
1  export const sum = (first, second) => {
2      return first + second
3  }
4
```

Membuat Unit Test

- Jest sudah menyediakan function yang diregistrasikan secara global bernama test(), function tersebut digunakan untuk membuat unit test
- test() memiliki parameter nama unit test dan juga function yang berisi kode unit test nya

Kode : Test Sum Function



The screenshot shows a code editor interface with a dark theme. On the left is a sidebar labeled "Project" containing a tree view of a project structure:

- belajar-nodejs-unit-test (highlighted)
- node_modules (library root)
- src
 - sum.js
- test
 - sum.test.js (highlighted)
 - package.json
 - package-lock.json
- External Libraries
- Scratches and Consoles

The main editor area shows the content of "sum.test.js":

```
import {sum} from "../src/sum";
test("sum(1, 2) must be 3", () => {
  const result = sum(1, 2)
  expect(result).toBe(3);
})
```

Hasil Test

```
SyntaxError: Cannot use import statement outside a module
```

```
    at Runtime.createScriptFromCode (node\_modules/jest-runtime/build/index.js:1773:14)
```

```
Test Suites: 1 failed, 1 total
```

```
Tests:       0 total
```

```
Snapshots:  0 total
```

```
Time:        0.362 s, estimated 1 s
```

```
Ran all test suites.
```

Pengenalan Babel

Kekurangan Jest

- Sejak awal belajar NodeJS, kita selalu menggunakan JavaScript Modules
- Sayangnya, Jest sampai dibuatnya materi ini, belum mendukung JavaScript Modules, masih menggunakan cara lama menggunakan CommonJS dengan memanfaatkan function require()
- Untungnya, ada library bernama Babel, yang bisa kita gunakan untuk membantu Jest

Jest Code Transformation

- Jest mendukung code transformation, yaitu fitur dimana kita bisa melakukan pengubahan kode program sebelum dijalankan oleh Jest
- Fitur ini yang dimanfaatkan untuk melakukan kompilasi ke kode JavaScript yang bisa dimengerti oleh Jest, memanfaatkan library Babel
- <https://jestjs.io/docs/code-transformation>



Babel

- Babel adalah JavaScript Compiler, yang digunakan untuk melakukan kompilasi kode JavaScript ke kode JavaScript yang berbeda versi, biasanya untuk ke versi yang lebih lama agar kompatibel dengan Browser versi lama
- Dengan Babel, kita bisa membuat kode program dengan fitur JavaScript terbaru, seperti Modules, tapi bisa di compile menjadi kode JavaScript lama sehingga compatible ketika dijalankan oleh teknologi lama atau yang belum mendukung fitur JavaScript baru
- <https://babeljs.io/>

Integrasi Babel dan Jest

- Jest terintegrasi dengan baik dengan Babel, sehingga Jest bisa secara otomatis melakukan kompilasi kode JavaScript unit test kita dengan Babel, dan menjalankan kode JavaScript dengan versi yang kompatibel dengan Jest
- <https://babeljs.io/setup>

Kode : Menjalankan Unit Test

```
→ belajar-nodejs-unit-test npm test
```

```
> belajar-nodejs-unit-test@1.0.0 test
> jest
```

```
PASS | test/sum.test.js
```

```
  ✓ sum 1 + 2 to equal 3 (2 ms)
```

```
Test Suites: 1 passed, 1 total
```

```
Tests:       1 passed, 1 total
```

```
Snapshots:  0 total
```

```
Time:        0.466 s
```

```
Ran all test suites.
```

Jest Configuration

Jest Configuration

- Jest memiliki banyak konfigurasi, namun jika kita tidak ubah konfigurasinya, Jest sudah memiliki default konfigurasi
- Ada banyak sekali konfigurasi yang terdapat di Jest, kita akan bahas sambil berjalan, dan yang memang diperlukan saja

File Konfigurasi Jest

- Jest sendiri mendukung dua cara untuk menyimpan data konfigurasi
- Pertama, menyimpan di file package.json dengan key jest
- Kedua dengan menyimpan sebagai file JavaScript di file jest.config.js/ts/mjs, atau membuatnya secara otomatis dengan perintah :
`jest --init`
- Jika menggunakan konfigurasi menggunakan file jest.config.js/ts/mjs, jangan lupa untuk memindahkan konfigurasi Jest di package.json

Pengaturan Konfigurasi

- Konfigurasi di Jest sangat sederhana, cukup gunakan key-value
- Dimana kita bisa melihat semua konfigurasi key yang tersedia dan kegunaannya di halaman <https://jestjs.io/docs/configuration>

Jest Command Line Interface

Jest Command Line Interface

- Seperti yang sudah dibahas sebelumnya, saat kita menginstall dependency Jest ke project kita, terdapat file jest di folder node_modules/.bin
- Dan untuk menjalankan unit test, kita bisa gunakan program jest
- Jest sendiri sebenarnya banyak sekali perintah tambahannya, dan kadang kita perlu memanfaatkannya, jadi tidak hanya mengetikkan perintah jest saja
- Untuk melihat detail perintah apa saja yang bisa kita gunakan, silahkan gunakan perintah :
jest --help

Kode : Run Test by Path

```
Terminal Local → belajar-nodejs-unit-test ./node_modules/.bin/jest --runTestsByPath test/sum.test.js
PASS  test/sum.test.js
  ✓ sum(1, 2) must be 3 (1 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.32 s, estimated 1 s
Ran all test suites within paths "test/sum.test.js".
→ belajar-nodejs-unit-test
```

Matchers

Matchers

- Saat kita membuat unit test, hal yang dilakukan adalah kita biasanya memiliki ekspektasi
- Contoh pada kode sum() sebelumnya, ketika kita panggil function sum() dengan parameter 1 dan 2, ekspektasi kita adalah hasil return dari function sum() tersebut adalah 3
- Di Jest, hal ini dinamakan Matchers
- <https://jestjs.io/docs/using-matchers>

Expect Function

- Matchers di Jest direpresentasikan dalam sebuah function bernama expect(value)
- Function expect() mengembalikan object Matchers, yang bisa kita gunakan untuk mengetest value yang kita expect()
- Ada banyak sekali function untuk melakukan test di Matchers, kita bisa baca detail nya di halaman dokumentasi API untuk function expect()
- <https://jestjs.io/docs/expect>

Equals Matchers

Equals Matchers

- Salah satu Matchers yang biasa digunakan ketika membuat unit test adalah equals matchers
- Ini digunakan untuk memastikan bahwa data sesuai atau sama dengan ekspektasi kita

Equals Matchers Functions

Function	Keterangan
expect(value).toBe(expected)	Value sama dengan expected, biasanya digunakan untuk value bukan object
expect(value).toEqual(expected)	Value sama dengan expected, dimana membandingkan semua properties secara recursive, atau dikenal dengan deep equality

Kode : Equals Matchers



The screenshot shows a code editor window with a dark theme. The file is named `equals.test.js`. It contains two test cases using the `test` function from the Jest library.

```
1  test("test toBe", () => {
2      let name = "Eko";
3      let hello = `Hello ${name}`;
4
5      expect(hello).toBe("Hello Eko");
6  })
7
8  test("test toEqual", () => {
9      let person = {id: "eko"};
10     Object.assign(person, {name: "Eko"})
11
12     expect(person).toEqual({id: "eko", name: "Eko"});
13  })
14 |
```

Truthiness Matchers

Truthiness Matchers

- Dalam unit test, kadang kita ingin membedakan antara undefined, null dan false.
- Dan kadang kita ingin melakukan ekspektasi nilai tersebut
- Jest memiliki matchers untuk melakukan hal tersebut juga

Truthiness Matchers Functions

Function	Keterangan
expect(value).toBeNull()	Memastikan value adalah null
expect(value).toBeUndefined()	Memastikan value adalah undefined
expect(value).toBeDefined()	Kebalikan dari toBeUndefined()
expect(value).toBeTruthy()	Memastikan value bernilai apapun, asal if statement menganggap true
expect(value).toBeFalsy()	Memastikan value bernilai apapun, asal if statement menganggap false



Kode : Truthiness Matchers

```
1 ► ⌂ test("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 = "Eko";
12  expect(value).toBeTruthy();
13});
14
```

Numbers Matchers

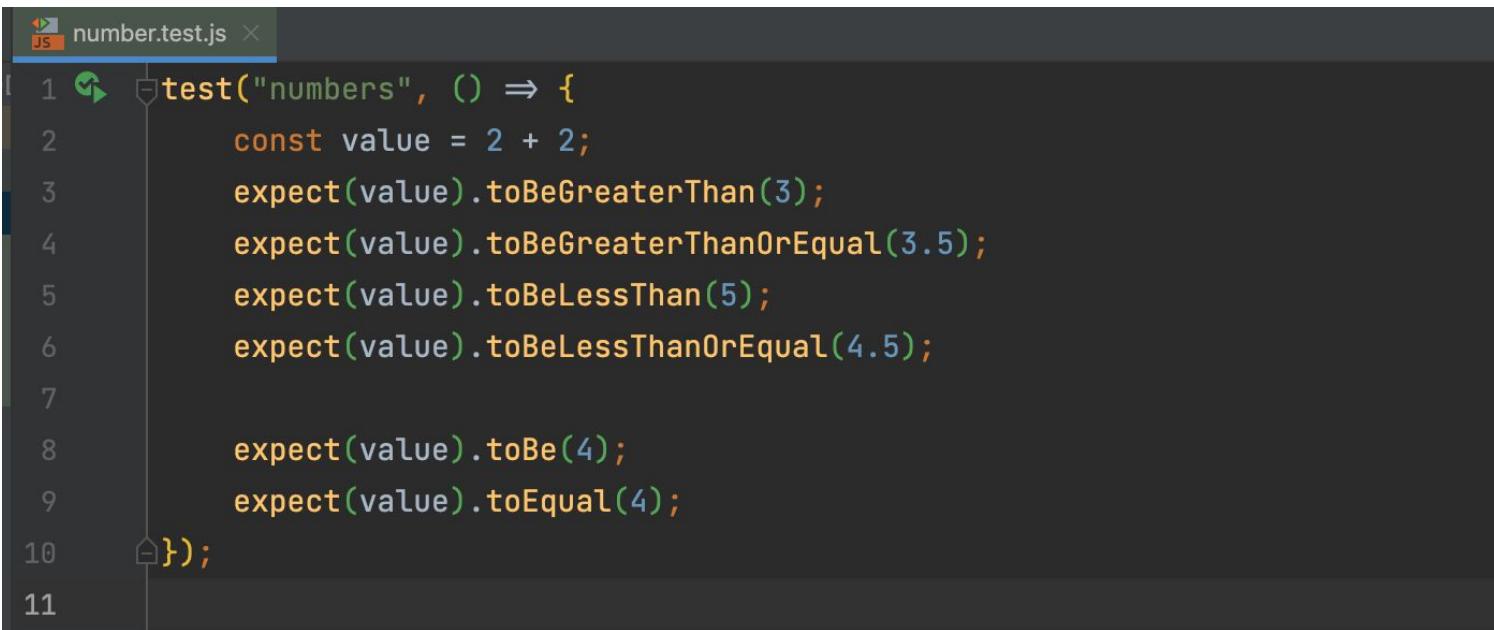
Numbers Matchers

- Jest juga memiliki matchers untuk digunakan untuk value berupa number
- Ketika value berupa number, kita juga bisa menggunakan toBe() dan toEqual(), untuk memastikan bahwa number bernilai sama dengan expected

Numbers Matchers Functions

Function	Keterangan
.toBeGreaterThan(n)	Memastikan value lebih besar dari n
.toBeGreaterThanOrEqual(n)	Memastikan value lebih besar atau sama dengan n
.toBeLessThan(n)	Memastikan value lebih kecil dari n
.toBeLessThanOrEqual(n)	Memastikan value lebih kecil atau sama dengan n

Kode : Numbers Matchers



A screenshot of a code editor showing a file named 'number.test.js'. The code uses Jest's number matchers to assert properties of a variable 'value'.

```
number.test.js
1 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
11});
```

Strings Matchers

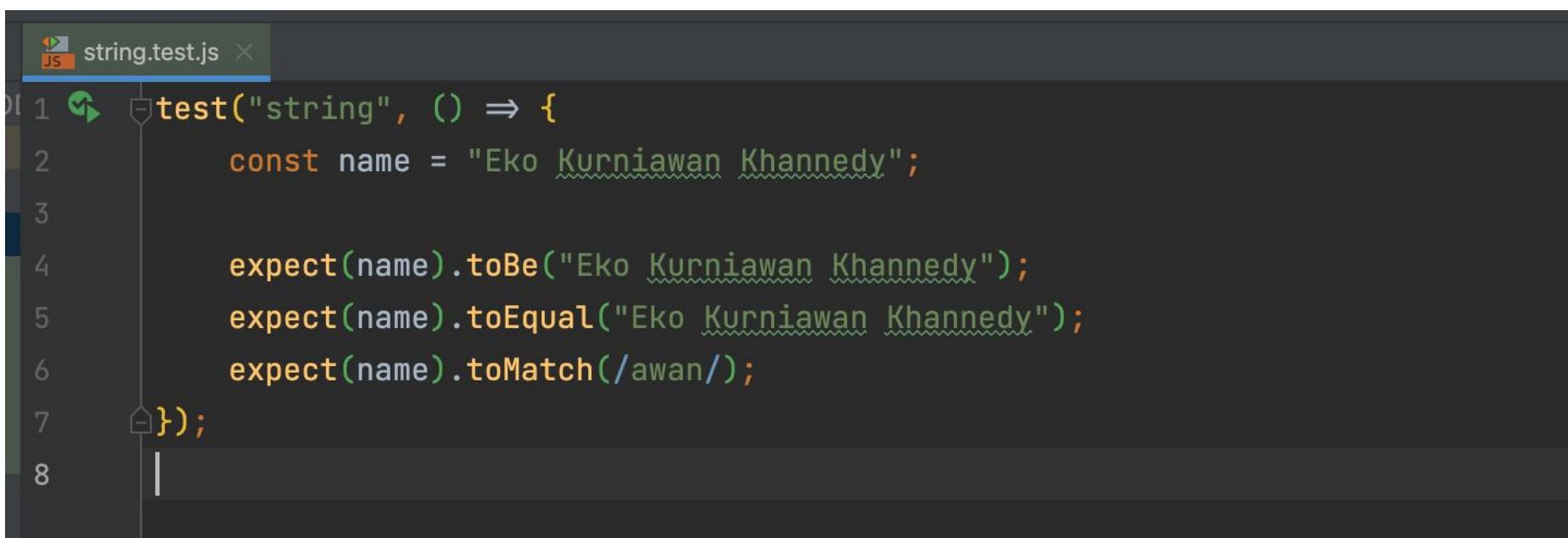
Strings Matchers

- Jest juga memiliki matchers function yang digunakan untuk value berupa String
- Jika kita ingin memastikan sebuah string sama, kita bisa gunakan toBe() atau toEqual()

Strings Matchers Functions

Function	Keterangan
.toMatch(regex)	Memastikan value sesuai dengan regex

Kode : Strings Matchers



```
string.test.js ×
1 1 test("string", () => {
2     const name = "Eko Kurniawan Khannedy";
3
4     expect(name).toBe("Eko Kurniawan Khannedy");
5     expect(name).toEqual("Eko Kurniawan Khannedy");
6     expect(name).toMatch(/awan/);
7 });
8 |
```

Arrays Matchers

Arrays Matchers

- Jest juga memiliki function yang bisa kita gunakan untuk mengecek data di dalam sebuah value array
- Jika ingin memastikan bahwa array sama, kita bisa menggunakan `toEqual()`

Arrays Matchers Functions

Function	Keterangan
.toContain(item)	Memastikan value array memiliki item, dimana pengecekan item menggunakan toBe()
.toContainEqual(item)	Memastikan value array memiliki item, dimana pengecekan item menggunakan toEqual()

Kode : Arrays Matchers

```
array.test.js ×
1  test("array", () => {
2    const names = ["Eko", "Kurniawan", "Khannedy"];
3    expect(names).toContain("Kurniawan");
4    expect(names).toEqual(["Eko", "Kurniawan", "Khannedy"]);
5
6    const persons = [{name: "Eko"}, {name: "Khannedy"}]
7    expect(persons).toContainEqual({name: "Khannedy"})
8    expect(persons).toEqual([{name: "Eko"}, {name: "Khannedy"}])
9  });
10
```

Exceptions Matchers

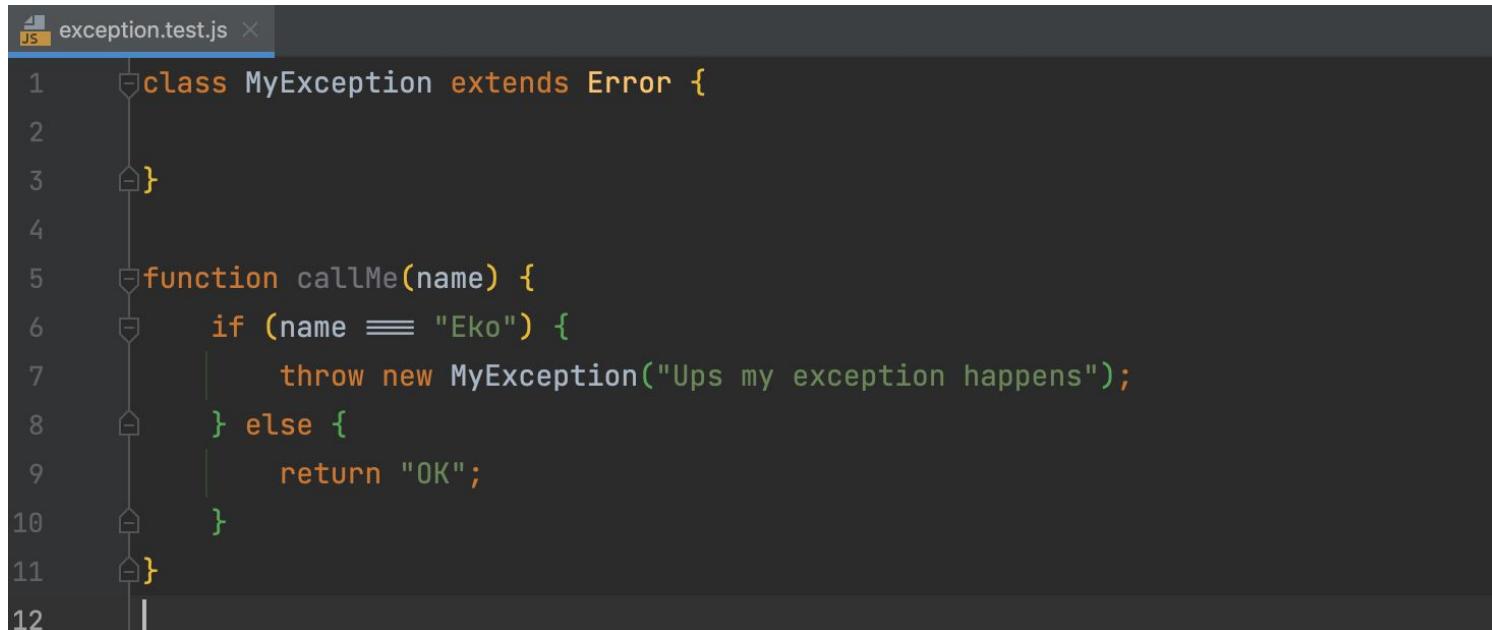
Exceptions Matchers

- Saat membuat kode program, kadang kita sering membuat exception
- Dalam unit test pun, kadang kita berharap sebuah exception terjadi
- Jest juga memiliki matchers untuk melakukan pengecekan exception
- Khusus untuk jenis matchers exception, kita perlu menggunakan closure function di value expect() nya, hal ini untuk memastikan exception ditangkap oleh matchers, jika tidak menggunakan closure function, maka exception akan terlanjur terjadi sebelum kita memanggil expect() function

Exceptions Matchers Functions

Function	Keterangan
.toThrow()	Memastikan terjadi exception apapun
.toThrow(exception)	Memastikan terjadi exception sesuai dengan expected exception
.toThrow(message)	Memastikan terjadi exception sesuai dengan string message

Kode : Contoh Exception dan Function



```
exception.test.js ×
1  class MyException extends Error {
2
3  }
4
5  function callMe(name) {
6    if (name === "Eko") {
7      throw new MyException("Ups my exception happens");
8    } else {
9      return "OK";
10   }
11 }
12 |
```



Kode : Exceptions Matchers

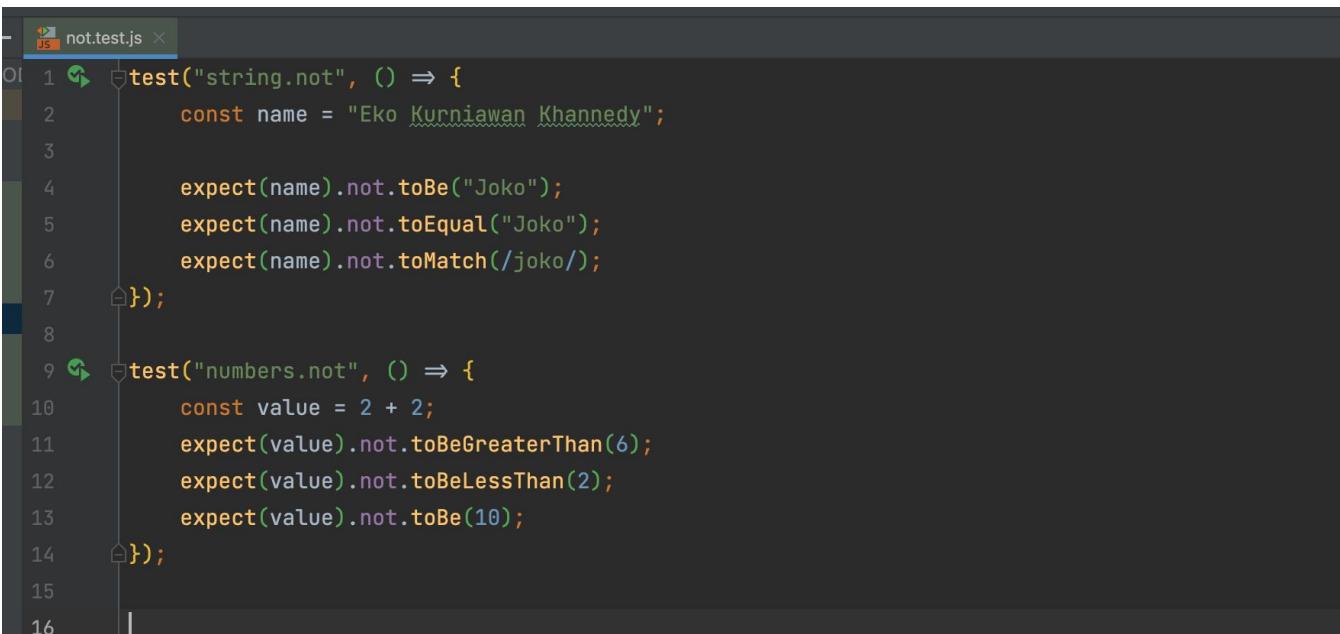
```
▶ test("exception", () => {
    expect(() => callMe("Eko")).toThrow();
    expect(() => callMe("Eko")).toThrow(MyException);
    expect(() => callMe("Eko")).toThrow("Ups my exception happens");
});
```

Not Matchers

Not Matchers

- Saat melakukan pengecekan menggunakan matchers, kadang-kadang kita ingin melakukan pengecekan kebalikannya
- Misal tidak sama dengan, tidak lebih dari, tidak contains, dan lain-lain
- Jest memiliki fitur untuk melakukan “not” di Matchers nya, dengan menggunakan property not di matchers, secara otomatis kita akan melakukan pengecekan kebalikannya
- Semua jenis matchers yang sudah kita bahas, mendukung property not ini

Kode : Not Matchers



The screenshot shows a code editor window with a dark theme. The file being edited is named 'not.test.js'. It contains two test cases, each using the Jest testing framework.

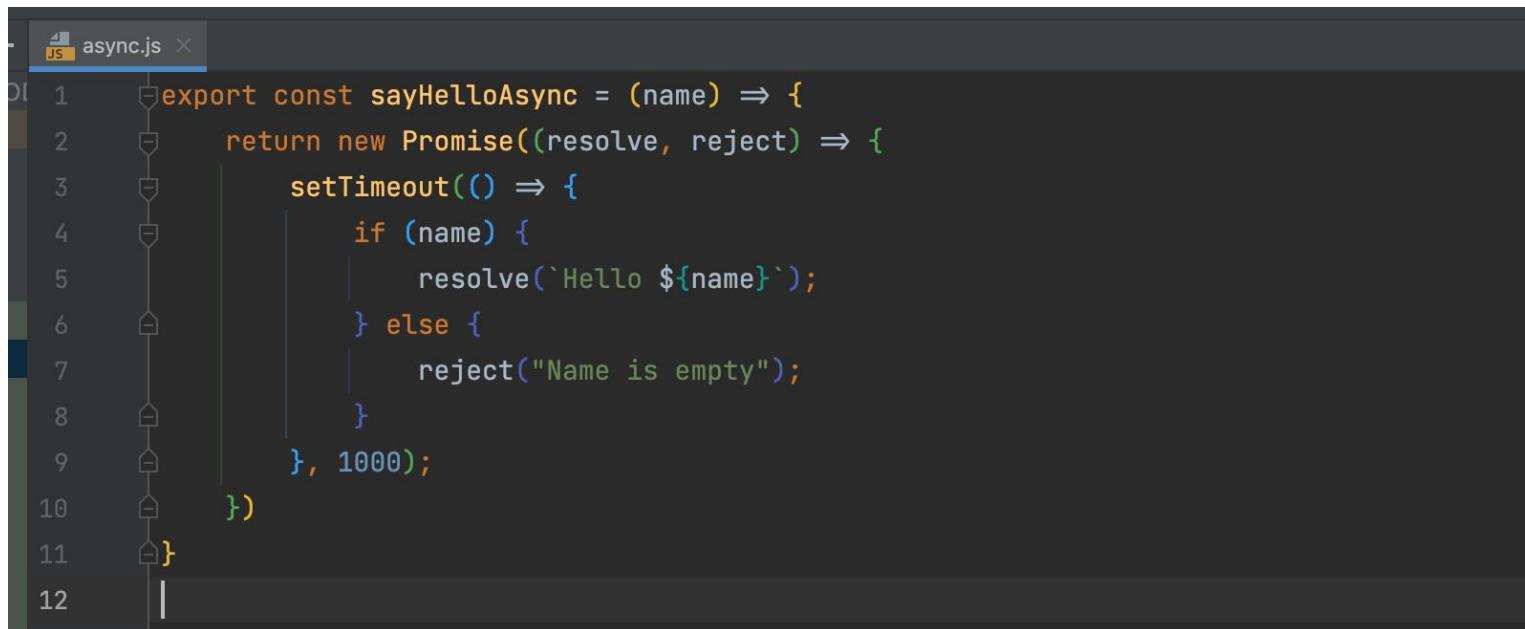
```
not.test.js
1 1 test("string.not", () => {
2     const name = "Eko Kurniawan Khannedy";
3
4     expect(name).not.toBe("Joko");
5     expect(name).not.toEqual("Joko");
6     expect(name).not.toMatch(/joko/);
7 });
8
9 2 test("numbers.not", () => {
10    const value = 2 + 2;
11    expect(value).not.toBeGreaterThan(6);
12    expect(value).not.toBeLessThan(2);
13    expect(value).not.toBe(10);
14 });
15
16
```

Test Async Code

Test Async Code

- Saat membuat kode program JavaScript, penggunaan kode asynchronous pasti sering kita gunakan, baik itu menggunakan Promise atau menggunakan Async Await
- Jest terintegrasi dengan baik jika kita ingin melakukan pengetesan terhadap kode yang async
- Namun saat kita melakukan pengetesan kode async, kita harus memberi tahu ke Jest, hal ini agar Jest tahu dan bisa menunggu kode async nya, sebelum melanjutkan ke unit test selanjutnya
- Caranya sebenarnya sangat mudah, kita cukup gunakan async code di closure function Jest

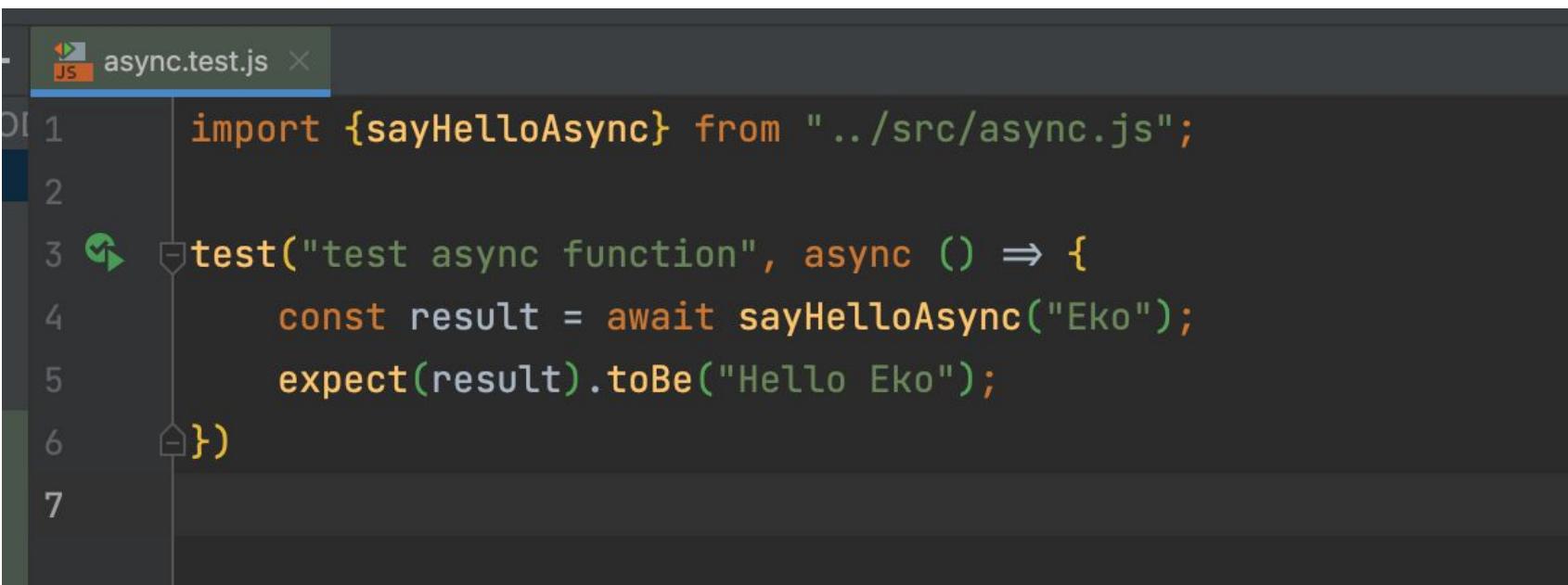
Kode : Async Function



The screenshot shows a code editor window with a dark theme. The file tab at the top is labeled "async.js". The code itself is an asynchronous function named "sayHelloAsync". It uses a Promise to delay the response by 1000 milliseconds. If a name is provided, it resolves with a greeting; if not, it rejects with an error message.

```
1  export const sayHelloAsync = (name) => {
2    return new Promise((resolve, reject) => {
3      setTimeout(() => {
4        if (name) {
5          resolve(`Hello ${name}`);
6        } else {
7          reject("Name is empty");
8        }
9      }, 1000);
10     })
11   }
```

Kode : Test Async Function



The screenshot shows a code editor window with a dark theme. The file being edited is named "async.test.js". The code inside the file is as follows:

```
1 import {sayHelloAsync} from "../src/async.js";
2
3 test("test async function", async () => {
4     const result = await sayHelloAsync("Eko");
5     expect(result).toBe("Hello Eko");
6 }
7 )
```

The code consists of seven numbered lines. Lines 1 through 6 are part of the test function body, while line 7 is the closing brace of the test function. A green checkmark icon is positioned next to the word "test" in line 3, indicating that the test has been run successfully.



Error : Regenerator Runtime

- Test suite failed to run

```
ReferenceError: regeneratorRuntime is not defined
```

```
1 | import {sayHelloAsync} from "../src/async.js";
2 |
> 3 | test("test async function", async () => {
|     ^
4 |     const result = await sayHelloAsync("Eko");
5 |     expect(result).toBe("Hello Eko");
6 | })
```

Regenerator Runtime Error

- Jangan khawatir jika terjadi regenerator runtime error.
- Ini adalah error yang terjadi di Babel, hal ini secara default Babel tidak melakukan memiliki fitur untuk melakukan kompilasi runtime ketika menemukan fitur regenerator atau async function
- Kita bisa menambahkan plugin untuk transform dan regenerator dengan menambahkan dependency:
`npm install @babel/plugin-transform-runtime`
- Selanjutnya tambahkan di `babel.config.json`

Kode : Babel Config



```
babel.config.json ×
1  {
2    "presets": [
3      "@babel/preset-env"
4    ],
5    "plugins": [
6      [
7        "@babel/plugin-transform-runtime",
8        {
9          "regenerator": true
10        }
11      ]
12    ]
13  }
```

Async Matchers

- Sebelumnya kita menggunakan async await untuk melakukan matchers, sebenarnya Jest juga memiliki fitur matchers terhadap data async atau Promise
- Hal ini mempermudah kita ketika ingin melakukan matchers, sehingga tidak perlu melakukan await pada async function nya
- Semua Async Matchers mengembalikan Promise

Async Matchers Functions

Function	Keterangan
expect(promise).resolves	Ekspektasi bahwa promise sukses, dan selanjutnya kita bisa gunakan Matchers function lainnya
expect(promise).rejects	Ekspektasi bahwa promise gagal, dan selanjutnya kita bisa gunakan Matchers function lainnya

Kode : Async Matchers Functions

```
test("test async matchers", async () => {
    await expect(sayHelloAsync("Eko")).resolves.toBe("Hello Eko");
    await expect(sayHelloAsync()).rejects.toBe("Name is empty");
});
```

Setup Function

Setup Function

- Kadang saat membuat unit test, kita membuat kode yang perlu dibuat sebelum unit test berjalan
- Selain itu, kadang kita juga kita membuat kode yang perlu dilakukan setelah unit test berjalan
- Jest memiliki fitur untuk menangani kasus seperti ini

Setup Functions

Function	Keterangan
beforeEach(function)	Function akan dieksekusi sebelum unit test berjalan, jika terdapat lima unit test dalam file, artinya akan dieksekusi juga sebanyak lima kali
afterEach(function)	Function akan dieksekusi setelah unit test selesai, jika terdapat lima unit test dalam file, artinya akan dieksekusi juga sebanyak lima kali

Kode : Setup Function

```
JS setup.test.js ×  
1 import {sum} from "../src/sum.js";  
2  
3 beforeEach(() => {  
4     console.info("Before Each");  
5});  
6  
7 afterEach(() => {  
8     console.info("After Each");  
9});  
10
```

```
10  
11 ↗  ⏷ test("first test", () => {  
12     expect(sum(10, 10)).toBe(20);  
13 } )  
14  
15 ↗  ⏷ test("second test", () => {  
16     expect(sum(10, 10)).toBe(20);  
17 } )  
18
```

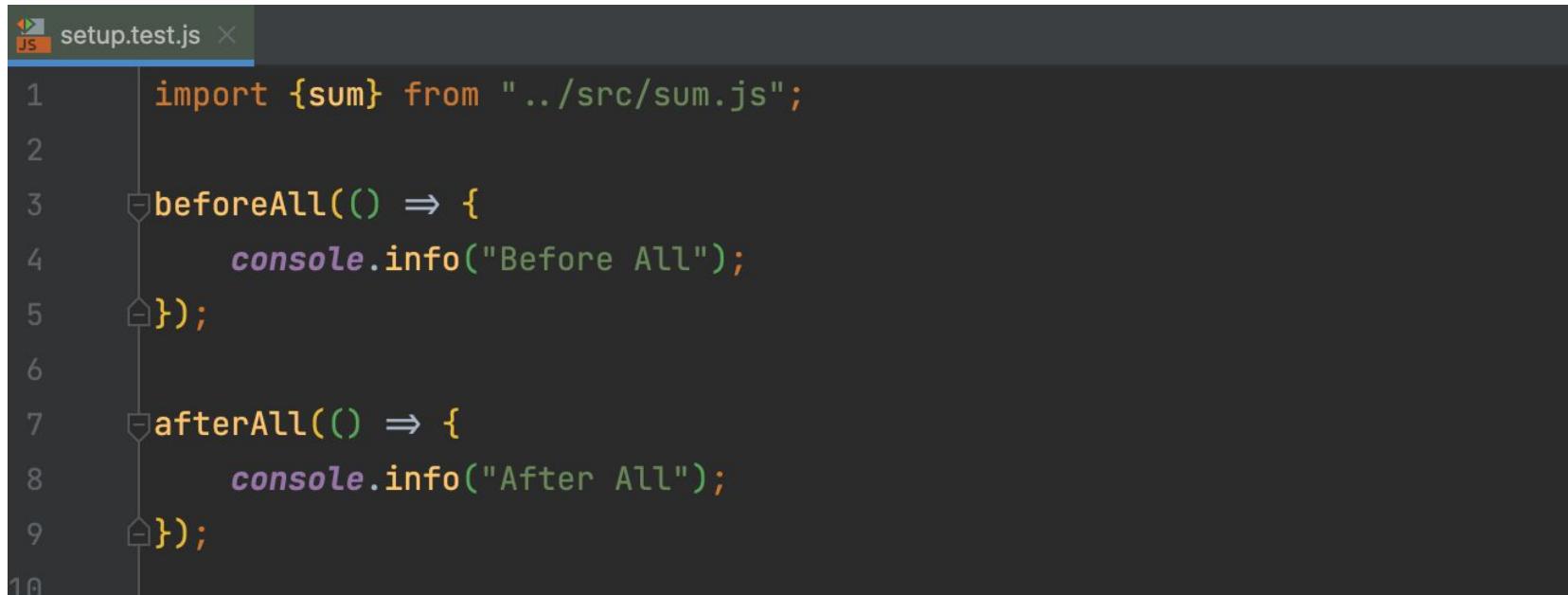
One-Time Setup Function

- Namun kadang-kadang, kita ingin membuat kode yang hanya dieksekusi sekali saja dalam sebuah file unit test
- Sekali sebelum semua unit test
- Dan sekali setelah semua unit test
- Jest juga menyediakan fitur tersebut

One-Time Setup Functions

Function	Keterangan
beforeAll(function)	Function akan dieksekusi sekali sebelum semua unit test berjalan di file unit test
afterAll(function)	Function akan dieksekusi sekali setelah semua unit test selesai di file unit test

Kode : One-Time Setup Function



A screenshot of a code editor showing a file named `setup.test.js`. The code contains two one-time setup functions: `beforeAll` and `afterAll`, both using the `console.info` method to log messages.

```
1 import {sum} from "../src/sum.js";
2
3 beforeAll(() => {
4     console.info("Before All");
5 });
6
7 afterAll(() => {
8     console.info("After All");
9 });
10
```

Async Setup Function

- Async juga bisa dilakukan di Setup Function
- Kita hanya perlu menggunakan kata kunci async pada parameter function di Setup Function nya

Scoping



Scoping

- Saat kita menggunakan Setup Function, secara default akan dieksekusi pada setiap test() function yang terdapat di file unit test
- Jest memiliki fitur scoping atau grouping, dimana kita bisa membuat group unit test menggunakan function describe()
- Setup Function yang dibuat di dalam describe() hanya digunakan untuk unit test di dalam describe() tersebut
- Namun Setup Function diluar describe() secara otomatis juga digunakan di dalam describe()

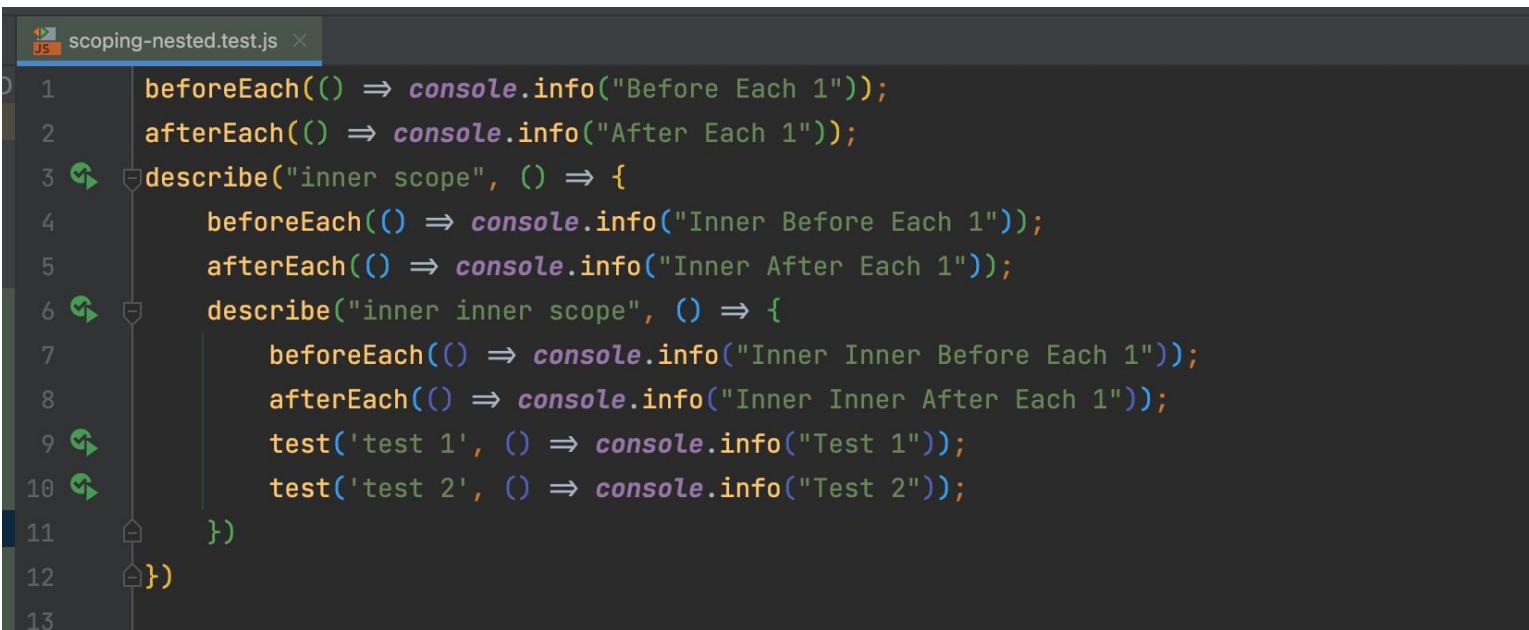
Kode : Scoping

```
1  beforeAll(() => console.info("Before All 1"));
2  afterAll(() => console.info("After All 1"));
3  beforeEach(() => console.info("Before Each 1"));
4  afterEach(() => console.info("After Each 1"));
5  test('test outer 1', () => console.info("Test Outer 1"));
6  test('test outer 2', () => console.info("Test Outer 2"));
7  describe("inner scope", () => {
8      beforeAll(() => console.info("Inner Before All 1"));
9      afterAll(() => console.info("Inner After All 1"));
10     beforeEach(() => console.info("Inner Before Each 1"));
11     afterEach(() => console.info("Inner After Each 1"));
12     test('test inner 1', () => console.info("Test Inner 1"));
13     test('test inner 2', () => console.info("Test Inner 2"));
14 })
```

Nested Scoping

- Jest juga mendukung nested scoping, artinya kita bisa membuat scoping menggunakan describe() function di dalam describe() function

Kode : Nested Scoping



The screenshot shows a code editor window with a dark theme. The file is named "scoping-nested.test.js". The code demonstrates nested scopes using Jest's test utilities. The code is as follows:

```
1  beforeEach(() => console.info("Before Each 1"));
2  afterEach(() => console.info("After Each 1"));
3  describe("inner scope", () => {
4      beforeEach(() => console.info("Inner Before Each 1"));
5      afterEach(() => console.info("Inner After Each 1"));
6      describe("inner inner scope", () => {
7          beforeEach(() => console.info("Inner Inner Before Each 1"));
8          afterEach(() => console.info("Inner Inner After Each 1"));
9          test('test 1', () => console.info("Test 1"));
10         test('test 2', () => console.info("Test 2"));
11     })
12 })
13 }
```

The code uses `console.info` to log messages at different levels of nesting. The first two lines are global setup. The third line starts a scope with a name ("inner scope"). Inside this scope, the fourth and fifth lines are setup. The sixth line starts another scope ("inner inner scope"). Inside this innermost scope, the seventh and eighth lines are setup. The ninth and tenth lines are test cases. The eleventh line ends the innermost scope, and the twelfth line ends the middle scope. The thirteenth line ends the global setup.

Code Coverage

Code Coverage

- Saat kita membuat unit test, kadang kita ingin tahu apakah semua kode kita sudah tercakupi dengan semua skenario unit test kita atau belum
- Jest memiliki fitur yang bernama Code Coverage, dengan ini, kita bisa melihat kode mana yang sudah tercakupi dengan unit test, dan mana yang belum
- Praktek ini merupakan salah satu best practice dengan menentukan jumlah persentase kode yang harus tercakupi oleh unit test, misal 80%

Menggunakan Fitur Code Coverage

- Secara default, Jest tidak menggunakan fitur Code Coverage, jika kita ingin menggunakan Code Coverage, kita perlu ubah konfigurasi Jest
- Caranya kita tambahkan atribut collectCoverage dengan nilai true
- <https://jestjs.io/docs/configuration#collectcoverage-boolean>

Kode : package.json

```
9      },
10     "jest": {
11       "transform": {
12         "^.+\\.[t|j]sx?$": "babel-jest"
13       },
14       "collectCoverage": true
15     },
16     "author": "Eko Kurniawan Khannedy",
```

Kode : Hasil Coverage

```
PASS test/async.test.js
```

File	%Stmts	%Branch	%Funcs	%Lines	Uncovered Line #s
All files	100	100	100	100	
async.js	100	100	100	100	
sum.js	100	100	100	100	

```
Test Suites: 12 passed, 12 total
```

```
Tests:      20 passed, 20 total
```

```
Snapshots: 0 total
```

```
Time:      5.404 s
```

```
Ran all test suites.
```

```
→ belajar-nodejs-unit-test []
```

Folder Coverage

- Jest Code Coverage secara otomatis membuat folder coverage di project kita
- Jangan lupa untuk meng-ignore folder tersebut agar tidak ter commit ke project kita
- Folder coverage tersebut berisi laporan Code Coverage berupa file html yang bisa kita lihat dengan mudah

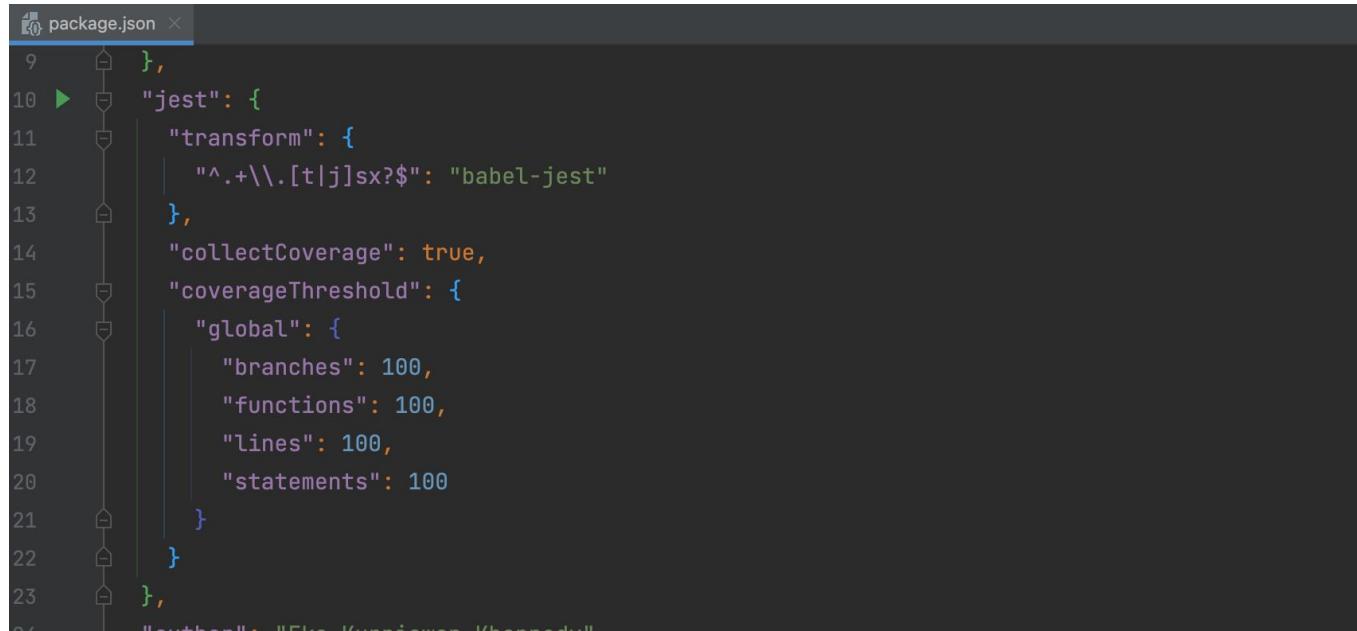
Coverage Threshold

- Kadang ada kalanya kita ingin memastikan persentase Code Coverage, hal ini agar programmer dalam project pasti membuat unit test dengan baik
- Jest memiliki fitur untuk menentukan Coverage Threshold dengan persentase, dimana jika Threshold nya dibawah persentase yang sudah ditentukan, secara otomatis maka unit test akan gagal
- Kita bisa tambahkan konfigurasi coverageThreshold
- <https://jestjs.io/docs/configuration#coverageThreshold-object>

Jenis Code Coverage

Jenis	Keterangan
branches	Alur program
functions	Function
lines	Baris
statements	Statement

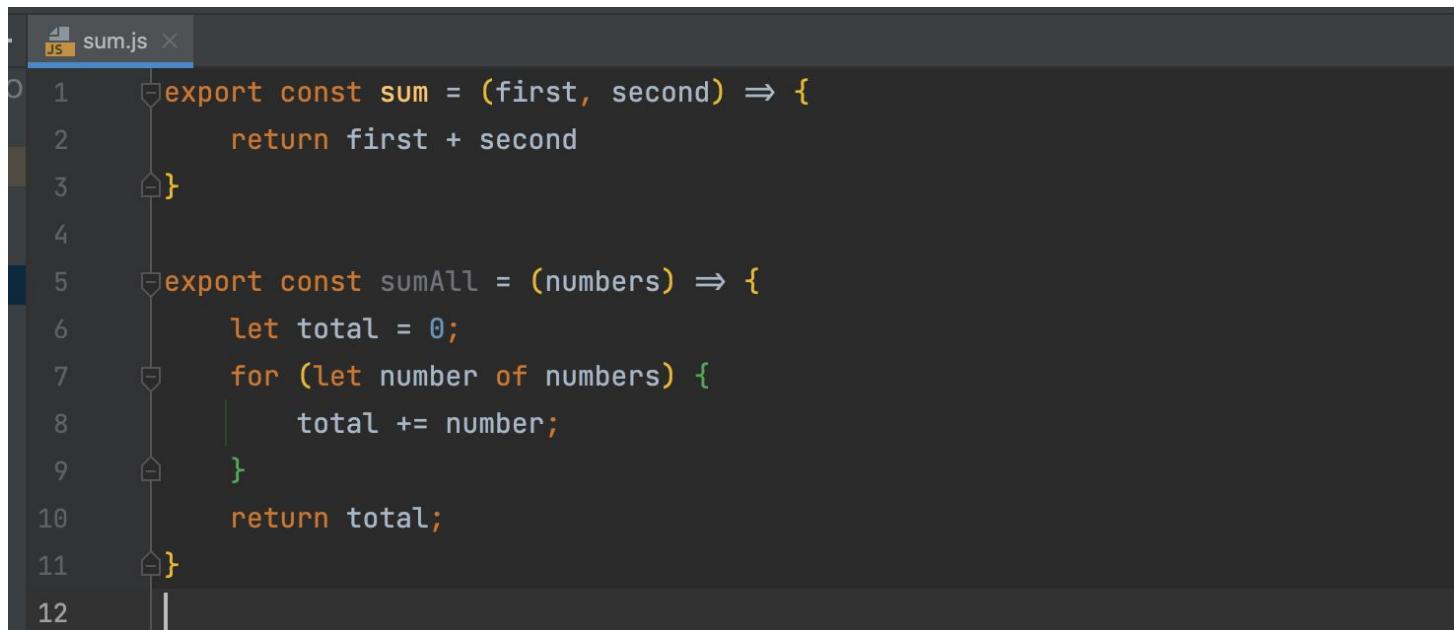
Kode : package.json



```
package.json ×
9   },
10  "jest": {
11    "transform": {
12      "^.+\\.[t|j]sx?$": "babel-jest"
13    },
14    "collectCoverage": true,
15    "coverageThreshold": {
16      "global": {
17        "branches": 100,
18        "functions": 100,
19        "lines": 100,
20        "statements": 100
21      }
22    }
23  },
```

The image shows a screenshot of a code editor displaying a `package.json` file. The file contains configuration for the `jest` testing framework. It includes a `transform` section for babel-jest, a `collectCoverage` option set to `true`, and a `coverageThreshold` section with a `global` object containing coverage thresholds for branches, functions, lines, and statements, all set to 100.

Kode : Update sum.js



```
sum.js
1  export const sum = (first, second) => {
2      return first + second
3  }
4
5  export const sumAll = (numbers) => {
6      let total = 0;
7      for (let number of numbers) {
8          total += number;
9      }
10     return total;
11 }
```

Kode : Hasil Unit Test

File	% Stmtts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	69.23	100	80	69.23	
async.js	100	100	100	100	
sum.js	42.85	100	50	42.85	6-10

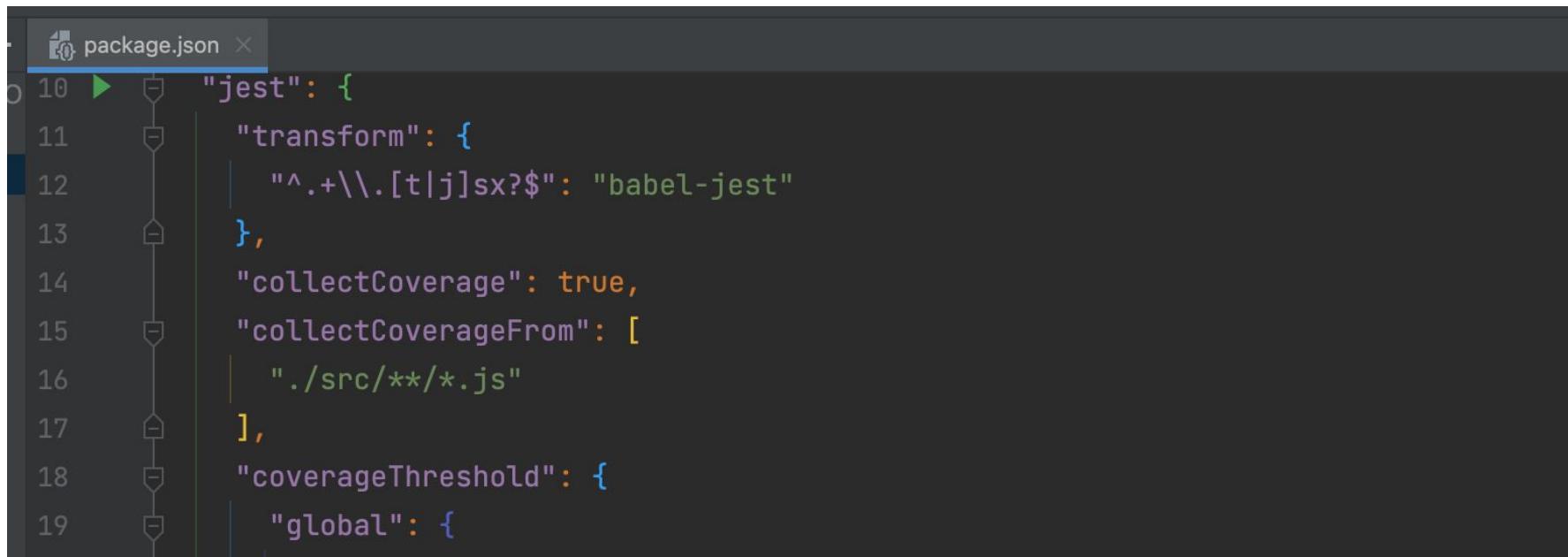
```
Jest: "global" coverage threshold for statements (100%) not met: 69.23%
Jest: "global" coverage threshold for lines (100%) not met: 69.23%
Jest: "global" coverage threshold for functions (100%) not met: 80%
```

```
Test Suites: 12 passed, 12 total
Tests:      20 passed, 20 total
Snapshots:  0 total
Time:      5.217 s
Ran all test suites.
```

Collect Coverage

- Kadang saat project kita sudah besar, kita ingin menentukan bagian kode mana yang ingin digunakan untuk menghitung Code Coverage nya
- Kita bisa menggunakan atribut collectCoverageFrom
- <https://jestjs.io/docs/configuration#collectcoveragefrom-array>

Kode : package.json



A screenshot of a code editor showing the `package.json` file. The file contains Jest configuration. The code is as follows:

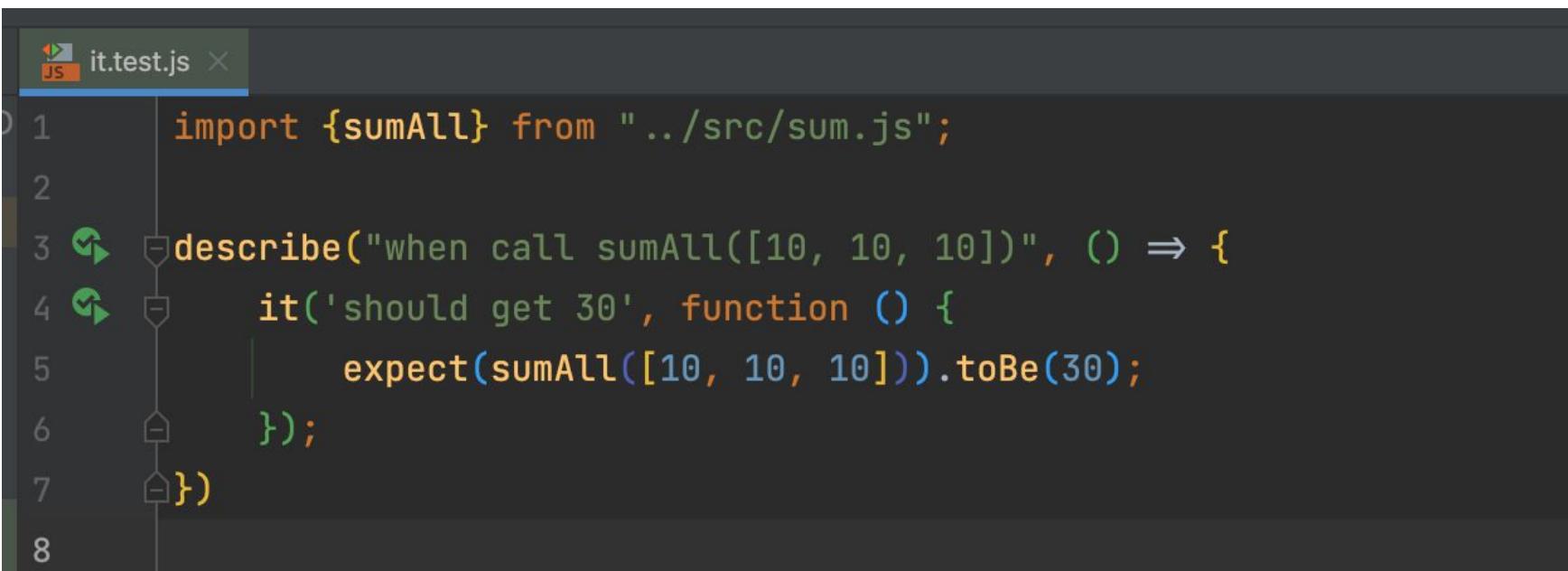
```
10  "jest": {  
11    "transform": {  
12      "^.+\\.[t|j]sx?$": "babel-jest"  
13    },  
14    "collectCoverage": true,  
15    "collectCoverageFrom": [  
16      "./src/**/*.{js,jsx}"  
17    ],  
18    "coverageThreshold": {  
19      "global": {  
20        "branches": 70,  
21        "functions": 70,  
22        "lines": 70,  
23        "statements": 70  
24      }  
25    }  
26  }  
27 }  
28 }  
29 }  
30 }  
31 }  
32 }  
33 }  
34 }  
35 }  
36 }  
37 }  
38 }  
39 }  
40 }  
41 }  
42 }  
43 }  
44 }  
45 }  
46 }  
47 }  
48 }  
49 }  
50 }  
51 }  
52 }  
53 }  
54 }  
55 }  
56 }  
57 }  
58 }  
59 }  
60 }  
61 }  
62 }  
63 }  
64 }  
65 }  
66 }  
67 }  
68 }  
69 }  
70 }  
71 }  
72 }  
73 }  
74 }  
75 }  
76 }  
77 }  
78 }  
79 }  
80 }  
81 }  
82 }  
83 }  
84 }  
85 }  
86 }  
87 }  
88 }  
89 }  
90 }  
91 }  
92 }  
93 }  
94 }  
95 }  
96 }  
97 }  
98 }  
99 }  
100 }  
101 }  
102 }  
103 }  
104 }  
105 }  
106 }  
107 }  
108 }  
109 }  
110 }  
111 }  
112 }  
113 }  
114 }  
115 }  
116 }  
117 }  
118 }  
119 }  
120 }  
121 }  
122 }  
123 }  
124 }  
125 }  
126 }  
127 }  
128 }  
129 }  
130 }  
131 }  
132 }  
133 }  
134 }  
135 }  
136 }  
137 }  
138 }  
139 }  
140 }  
141 }  
142 }  
143 }  
144 }  
145 }  
146 }  
147 }  
148 }  
149 }  
150 }  
151 }  
152 }  
153 }  
154 }  
155 }  
156 }  
157 }  
158 }  
159 }  
160 }  
161 }  
162 }  
163 }  
164 }  
165 }  
166 }  
167 }  
168 }  
169 }  
170 }  
171 }  
172 }  
173 }  
174 }  
175 }  
176 }  
177 }  
178 }  
179 }  
180 }  
181 }  
182 }  
183 }  
184 }  
185 }  
186 }  
187 }  
188 }  
189 }  
190 }  
191 }  
192 }  
193 }  
194 }  
195 }  
196 }  
197 }  
198 }  
199 }  
200 }  
201 }  
202 }  
203 }  
204 }  
205 }  
206 }  
207 }  
208 }  
209 }  
210 }  
211 }  
212 }  
213 }  
214 }  
215 }  
216 }  
217 }  
218 }  
219 }  
220 }  
221 }  
222 }  
223 }  
224 }  
225 }  
226 }  
227 }  
228 }  
229 }  
230 }  
231 }  
232 }  
233 }  
234 }  
235 }  
236 }  
237 }  
238 }  
239 }  
240 }  
241 }  
242 }  
243 }  
244 }  
245 }  
246 }  
247 }  
248 }  
249 }  
250 }  
251 }  
252 }  
253 }  
254 }  
255 }  
256 }  
257 }  
258 }  
259 }  
260 }  
261 }  
262 }  
263 }  
264 }  
265 }  
266 }  
267 }  
268 }  
269 }  
270 }  
271 }  
272 }  
273 }  
274 }  
275 }  
276 }  
277 }  
278 }  
279 }  
280 }  
281 }  
282 }  
283 }  
284 }  
285 }  
286 }  
287 }  
288 }  
289 }  
290 }  
291 }  
292 }  
293 }  
294 }  
295 }  
296 }  
297 }  
298 }  
299 }  
300 }  
301 }  
302 }  
303 }  
304 }  
305 }  
306 }  
307 }  
308 }  
309 }  
310 }  
311 }  
312 }  
313 }  
314 }  
315 }  
316 }  
317 }  
318 }  
319 }  
320 }  
321 }  
322 }  
323 }  
324 }  
325 }  
326 }  
327 }  
328 }  
329 }  
330 }  
331 }  
332 }  
333 }  
334 }  
335 }  
336 }  
337 }  
338 }  
339 }  
340 }  
341 }  
342 }  
343 }  
344 }  
345 }  
346 }  
347 }  
348 }  
349 }  
350 }  
351 }  
352 }  
353 }  
354 }  
355 }  
356 }  
357 }  
358 }  
359 }  
360 }  
361 }  
362 }  
363 }  
364 }  
365 }  
366 }  
367 }  
368 }  
369 }  
370 }  
371 }  
372 }  
373 }  
374 }  
375 }  
376 }  
377 }  
378 }  
379 }  
380 }  
381 }  
382 }  
383 }  
384 }  
385 }  
386 }  
387 }  
388 }  
389 }  
390 }  
391 }  
392 }  
393 }  
394 }  
395 }  
396 }  
397 }  
398 }  
399 }  
400 }  
401 }  
402 }  
403 }  
404 }  
405 }  
406 }  
407 }  
408 }  
409 }  
410 }  
411 }  
412 }  
413 }  
414 }  
415 }  
416 }  
417 }  
418 }  
419 }  
420 }  
421 }  
422 }  
423 }  
424 }  
425 }  
426 }  
427 }  
428 }  
429 }  
430 }  
431 }  
432 }  
433 }  
434 }  
435 }  
436 }  
437 }  
438 }  
439 }  
440 }  
441 }  
442 }  
443 }  
444 }  
445 }  
446 }  
447 }  
448 }  
449 }  
450 }  
451 }  
452 }  
453 }  
454 }  
455 }  
456 }  
457 }  
458 }  
459 }  
460 }  
461 }  
462 }  
463 }  
464 }  
465 }  
466 }  
467 }  
468 }  
469 }  
470 }  
471 }  
472 }  
473 }  
474 }  
475 }  
476 }  
477 }  
478 }  
479 }  
480 }  
481 }  
482 }  
483 }  
484 }  
485 }  
486 }  
487 }  
488 }  
489 }  
490 }  
491 }  
492 }  
493 }  
494 }  
495 }  
496 }  
497 }  
498 }  
499 }  
500 }  
501 }  
502 }  
503 }  
504 }  
505 }  
506 }  
507 }  
508 }  
509 }  
510 }  
511 }  
512 }  
513 }  
514 }  
515 }  
516 }  
517 }  
518 }  
519 }  
520 }  
521 }  
522 }  
523 }  
524 }  
525 }  
526 }  
527 }  
528 }  
529 }  
530 }  
531 }  
532 }  
533 }  
534 }  
535 }  
536 }  
537 }  
538 }  
539 }  
540 }  
541 }  
542 }  
543 }  
544 }  
545 }  
546 }  
547 }  
548 }  
549 }  
550 }  
551 }  
552 }  
553 }  
554 }  
555 }  
556 }  
557 }  
558 }  
559 }  
560 }  
561 }  
562 }  
563 }  
564 }  
565 }  
566 }  
567 }  
568 }  
569 }  
570 }  
571 }  
572 }  
573 }  
574 }  
575 }  
576 }  
577 }  
578 }  
579 }  
580 }  
581 }  
582 }  
583 }  
584 }  
585 }  
586 }  
587 }  
588 }  
589 }  
590 }  
591 }  
592 }  
593 }  
594 }  
595 }  
596 }  
597 }  
598 }  
599 }  
600 }  
601 }  
602 }  
603 }  
604 }  
605 }  
606 }  
607 }  
608 }  
609 }  
610 }  
611 }  
612 }  
613 }  
614 }  
615 }  
616 }  
617 }  
618 }  
619 }  
620 }  
621 }  
622 }  
623 }  
624 }  
625 }  
626 }  
627 }  
628 }  
629 }  
630 }  
631 }  
632 }  
633 }  
634 }  
635 }  
636 }  
637 }  
638 }  
639 }  
640 }  
641 }  
642 }  
643 }  
644 }  
645 }  
646 }  
647 }  
648 }  
649 }  
650 }  
651 }  
652 }  
653 }  
654 }  
655 }  
656 }  
657 }  
658 }  
659 }  
660 }  
661 }  
662 }  
663 }  
664 }  
665 }  
666 }  
667 }  
668 }  
669 }  
670 }  
671 }  
672 }  
673 }  
674 }  
675 }  
676 }  
677 }  
678 }  
679 }  
680 }  
681 }  
682 }  
683 }  
684 }  
685 }  
686 }  
687 }  
688 }  
689 }  
690 }  
691 }  
692 }  
693 }  
694 }  
695 }  
696 }  
697 }  
698 }  
699 }  
700 }  
701 }  
702 }  
703 }  
704 }  
705 }  
706 }  
707 }  
708 }  
709 }  
710 }  
711 }  
712 }  
713 }  
714 }  
715 }  
716 }  
717 }  
718 }  
719 }  
720 }  
721 }  
722 }  
723 }  
724 }  
725 }  
726 }  
727 }  
728 }  
729 }  
730 }  
731 }  
732 }  
733 }  
734 }  
735 }  
736 }  
737 }  
738 }  
739 }  
740 }  
741 }  
742 }  
743 }  
744 }  
745 }  
746 }  
747 }  
748 }  
749 }  
750 }  
751 }  
752 }  
753 }  
754 }  
755 }  
756 }  
757 }  
758 }  
759 }  
760 }  
761 }  
762 }  
763 }  
764 }  
765 }  
766 }  
767 }  
768 }  
769 }  
770 }  
771 }  
772 }  
773 }  
774 }  
775 }  
776 }  
777 }  
778 }  
779 }  
780 }  
781 }  
782 }  
783 }  
784 }  
785 }  
786 }  
787 }  
788 }  
789 }  
790 }  
791 }  
792 }  
793 }  
794 }  
795 }  
796 }  
797 }  
798 }  
799 }  
800 }  
801 }  
802 }  
803 }  
804 }  
805 }  
806 }  
807 }  
808 }  
809 }  
810 }  
811 }  
812 }  
813 }  
814 }  
815 }  
816 }  
817 }  
818 }  
819 }  
820 }  
821 }  
822 }  
823 }  
824 }  
825 }  
826 }  
827 }  
828 }  
829 }  
830 }  
831 }  
832 }  
833 }  
834 }  
835 }  
836 }  
837 }  
838 }  
839 }  
840 }  
841 }  
842 }  
843 }  
844 }  
845 }  
846 }  
847 }  
848 }  
849 }  
850 }  
851 }  
852 }  
853 }  
854 }  
855 }  
856 }  
857 }  
858 }  
859 }  
860 }  
861 }  
862 }  
863 }  
864 }  
865 }  
866 }  
867 }  
868 }  
869 }  
870 }  
871 }  
872 }  
873 }  
874 }  
875 }  
876 }  
877 }  
878 }  
879 }  
880 }  
881 }  
882 }  
883 }  
884 }  
885 }  
886 }  
887 }  
888 }  
889 }  
890 }  
891 }  
892 }  
893 }  
894 }  
895 }  
896 }  
897 }  
898 }  
899 }  
900 }  
901 }  
902 }  
903 }  
904 }  
905 }  
906 }  
907 }  
908 }  
909 }  
910 }  
911 }  
912 }  
913 }  
914 }  
915 }  
916 }  
917 }  
918 }  
919 }  
920 }  
921 }  
922 }  
923 }  
924 }  
925 }  
926 }  
927 }  
928 }  
929 }  
930 }  
931 }  
932 }  
933 }  
934 }  
935 }  
936 }  
937 }  
938 }  
939 }  
940 }  
941 }  
942 }  
943 }  
944 }  
945 }  
946 }  
947 }  
948 }  
949 }  
950 }  
951 }  
952 }  
953 }  
954 }  
955 }  
956 }  
957 }  
958 }  
959 }  
960 }  
961 }  
962 }  
963 }  
964 }  
965 }  
966 }  
967 }  
968 }  
969 }  
970 }  
971 }  
972 }  
973 }  
974 }  
975 }  
976 }  
977 }  
978 }  
979 }  
980 }  
981 }  
982 }  
983 }  
984 }  
985 }  
986 }  
987 }  
988 }  
989 }  
990 }  
991 }  
992 }  
993 }  
994 }  
995 }  
996 }  
997 }  
998 }  
999 }  
1000 }
```

It Function

It Function

- Sebelumnya untuk membuat unit test, kita menggunakan function test()
- Di Jest, terdapat alias untuk function test(), yaitu it()
- Sebenarnya tidak ada bedanya dengan function test(), hanya saja, kadang ada programmer yang lebih suka menggunakan function it() agar unit test yang dibuat mirip dengan cerita ketika dibaca kodennya

Kode : It Function



```
it.test.js ×

1 import {sumAll} from "../src/sum.js";
2
3 describe("when call sumAll([10, 10, 10])", () => {
4   it('should get 30', function () {
5     expect(sumAll([10, 10, 10])).toBe(30);
6   });
7 });
8 }
```

Skip Function

Skip Function

- Saat membuat unit test, lalu kita mendapatkan masalah di salah satu unit test, kadang kita ingin meng-ignore unit test tersebut terlebih dahulu
- Kita tidak perlu menambahkan komentar pada unit test tersebut
- Kita bisa menggunakan skip function, yang secara otomatis akan menjadikan unit test tersebut ter-ignore dan tidak akan di eksekusi
- <https://jestjs.io/docs/api#testskipname-fn>

Kode : Skip Function



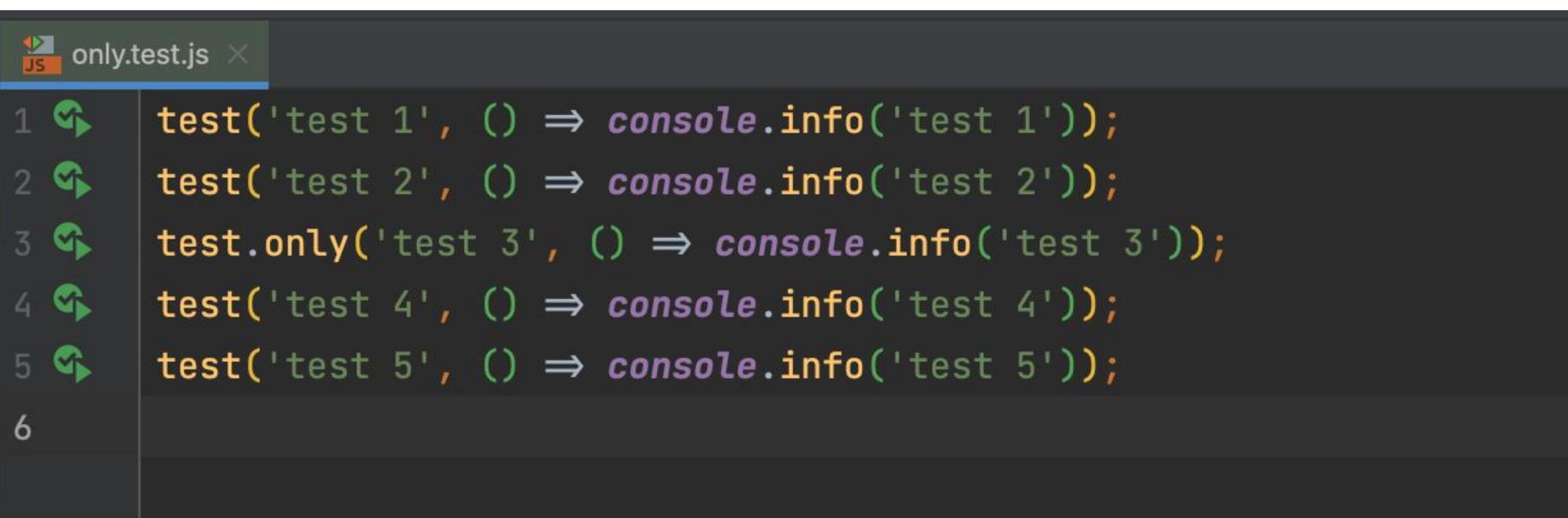
```
skip.test.js ×
1  ↗  test('test 1', () => console.info('test 1'));
2  ↗  test('test 2', () => console.info('test 2'));
3  ▶  test.skip('test 3', () => console.info('test 3'));
4  ↗  test('test 4', () => console.info('test 4'));
5  ↗  test('test 5', () => console.info('test 5'));
6
```

Only Function

Only Function

- Ketika kita melakukan proses debugging di unit test di dalam sebuah file yang unit test nya banyak, kadang kita ingin fokus ke unit test tertentu
- Jika kita menggunakan skip unit test yang lain, maka akan sulit jika terlalu banyak
- Kita bisa menggunakan Only Function, untuk memaksa dalam file tersebut, hanya unit test yang ditandai dengan Only yang di eksekusi
- <https://jestjs.io/docs/api#testonlyname-fn-timeout>

Kode : Only Function



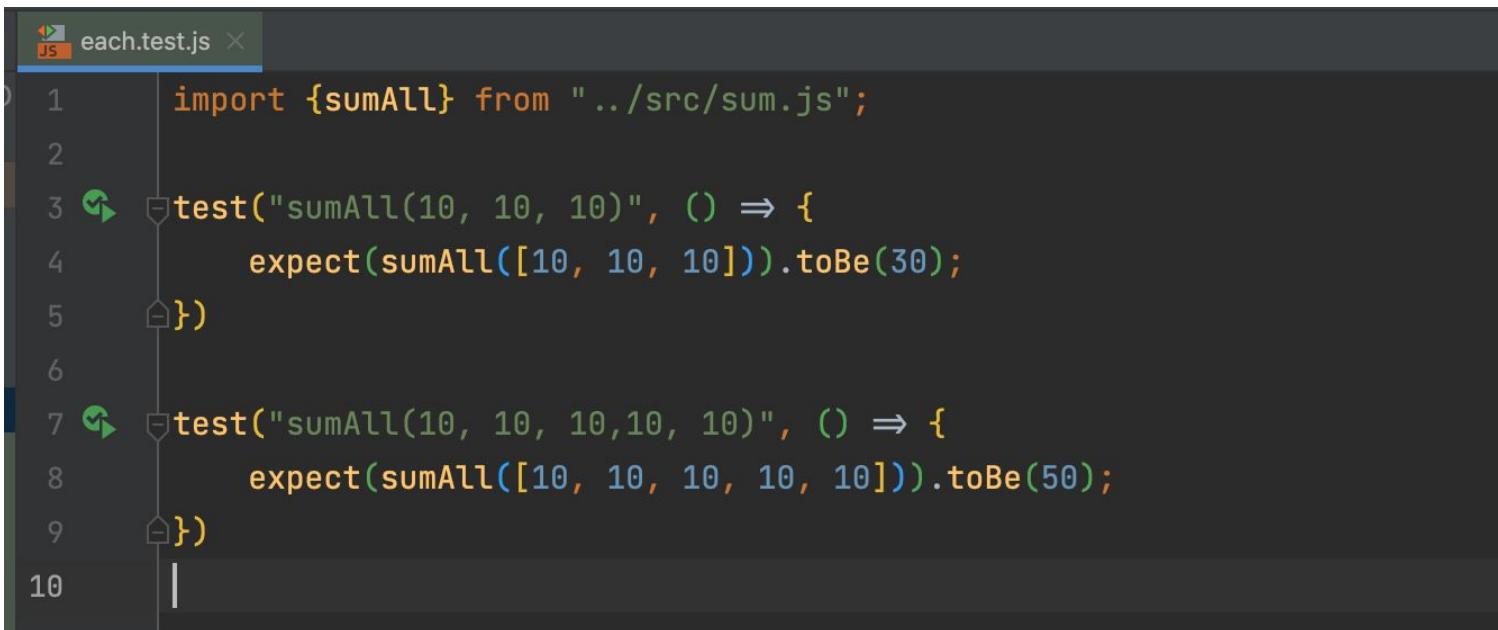
```
only.test.js
1 1 ↵ test('test 1', () => console.info('test 1'));
2 2 ↵ test('test 2', () => console.info('test 2'));
3 3 ↵ test.only('test 3', () => console.info('test 3'));
4 4 ↵ test('test 4', () => console.info('test 4'));
5 5 ↵ test('test 5', () => console.info('test 5'));
6
```

Each Function

Duplicate Unit Test

- Salah satu kesalahan yang biasa dilakukan adalah membuat unit test yang duplicate
- Biasanya alasan melakukan duplicate unit test, hanya karena data yang di test nya saja berbeda

Kode : Duplicate Unit Test



The screenshot shows a code editor window with a dark theme. The file tab at the top is labeled "each.test.js". The code itself is as follows:

```
1 import {sumAll} from "../src/sum.js";
2
3 test("sumAll(10, 10, 10)", () => {
4     expect(sumAll([10, 10, 10])).toBe(30);
5 })
6
7 test("sumAll(10, 10, 10,10, 10)", () => {
8     expect(sumAll([10, 10, 10, 10, 10])).toBe(50);
9 })
10
```

The code consists of two test cases for the `sumAll` function. Both tests pass, as indicated by the green checkmarks next to the test names. The second test case is identical to the first but with an additional argument value of 10.

Each Function

- Pada kasus seperti ini, dimana kode unit test nya tidak berbeda, yang berbeda hanya datanya saja, sangat direkomendasikan menggunakan Each Function di Jest
- Each Function memungkinkan kita menggunakan data dalam bentuk array, yang akan di iterasi ke dalam kode unit test yang sama
- <https://jestjs.io/docs/api#testeachtablename-fn-timeout>



Kode : Each Function

```
JS each.test.js ×  
JS each.test.js > ...  
1 import {sumAll} from "../src/sum.js";  
2  
3 const table = [  
4   [[10, 10, 10], 30],  
5   [[10, 10, 10, 10, 10], 50],  
6   [[10, 10, 10, 10, 10, 10, 10], 70],  
7 ];  
8  
9 test.each(table)("test sumAll(%s) should result %i", (numbers, expected) => {  
10   expect(sumAll(numbers)).toBe(expected);  
11 })  
12  
13
```

Kode : Hasil Unit Test

```
→ belajar-nodejs-unit-test npx jest test/each.test.js
PASS  test/each.test.js
  ✓ test sumAll([ 10, 10, 10 ]) should result 30 (2 ms)
  ✓ test sumAll([ 10, 10, 10, 10, 10 ]) should result 50 (1 ms)
  ✓ test sumAll([
    10, 10, 10, 10,
    10, 10, 10
  ]) should result 70
```

Object Sebagai Data

- Kadang, saat menggunakan data Array, jika terlalu banyak parameternya, maka akan membingungkan
- Each Function juga bisa menggunakan data Object, namun kita perlu melakukan destructuring



Kode : Each Function dengan Object

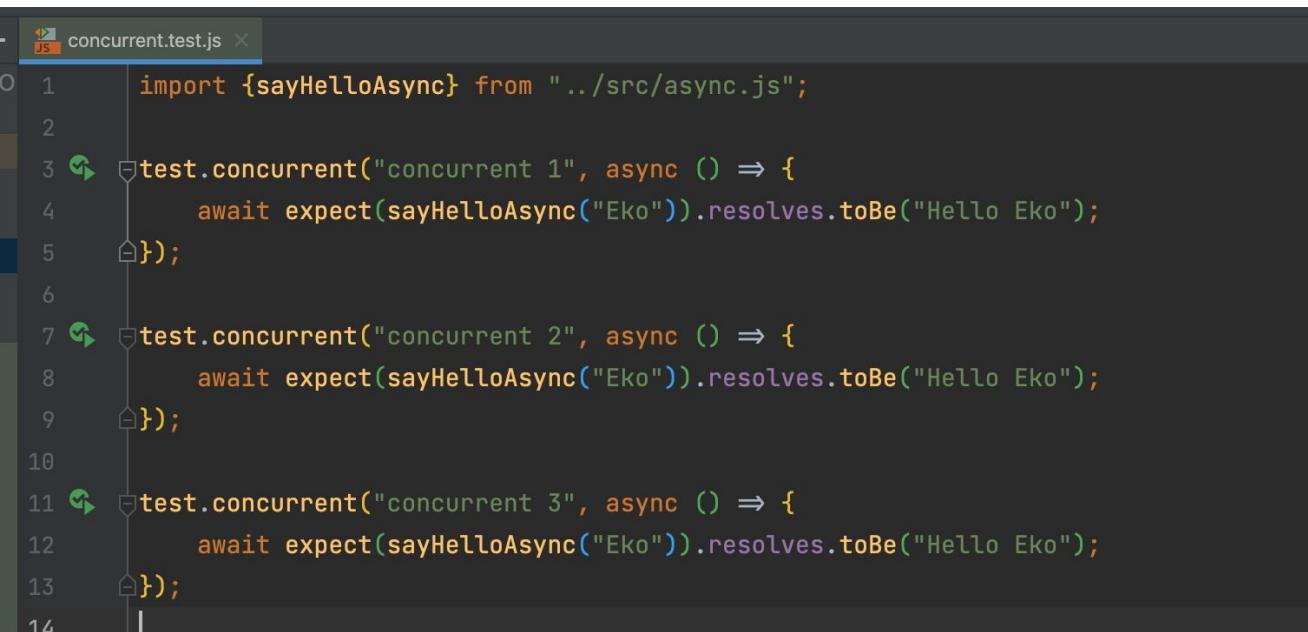
```
JS each.test.js ×  
JS each.test.js > ...  
1 import {sumAll} from "../src/sum.js";  
2  
3 const table = [  
4   {numbers: [10, 10, 10], expected: 30},  
5   {numbers: [10, 10, 10, 10, 10], expected: 50},  
6   {numbers: [10, 10, 10, 10, 10, 10, 10], expected: 70},  
7 ];  
8  
9 test.each(table)("test sumAll(%s) should result %i", ({numbers, expected}) => {  
10   expect(sumAll(numbers)).toBe(expected);  
11 })  
12  
13 |
```

Concurrent Test

Concurrent Test

- Secara default, semua unit test akan dijalankan secara sequential, dan unit test selanjutnya akan dijalankan ketika unit test sebelumnya telah selesai
- Jest juga mendukung concurrent unit test, dimana kita bisa menandai sebuah unit test agar jalannya secara concurrent atau async sehingga tidak perlu ditunggu
- Saat dibuatnya materi ini, fitur concurrent sendiri masih experimental atau belum stabil
- <https://jestjs.io/docs/api#testconcurrentname-fn-timeout>

Kode : Concurrent Test



```
concurrent.test.js ×
0 1 import {sayHelloAsync} from "../src/async.js";
2
3 ⚡ test.concurrent("concurrent 1", async () => {
4     await expect(sayHelloAsync("Eko")).resolves.toBe("Hello Eko");
5 });
6
7 ⚡ test.concurrent("concurrent 2", async () => {
8     await expect(sayHelloAsync("Eko")).resolves.toBe("Hello Eko");
9 });
10
11 ⚡ test.concurrent("concurrent 3", async () => {
12     await expect(sayHelloAsync("Eko")).resolves.toBe("Hello Eko");
13 });
14
```

Membatasi Concurrent

- Kita bisa membatasi berapa banyak concurrent test yang berjalan dengan cara menambahkan konfigurasi di Jest nya
- <https://jestjs.io/docs/configuration#maxconcurrency-number>

Kode : package.json

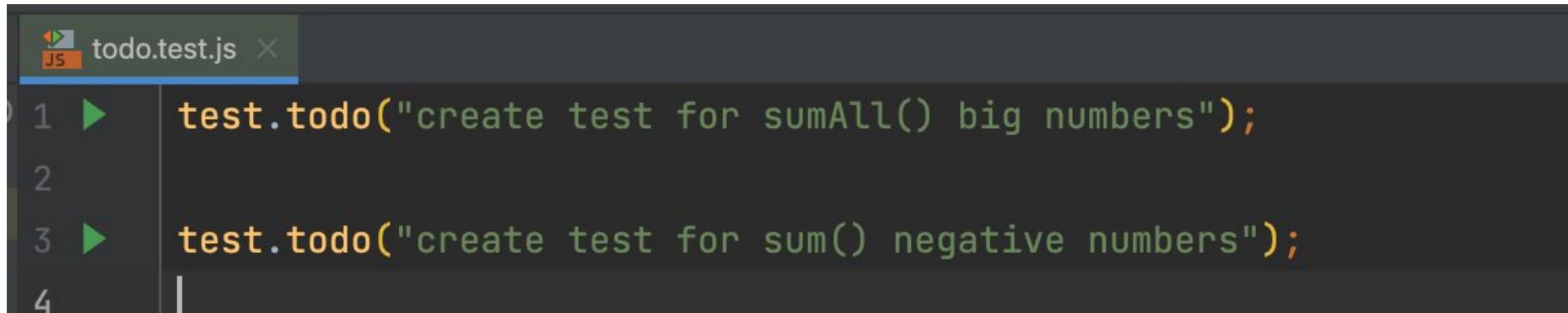
```
package.json ×
0   "type": "module",
1
2   "scripts": {
3     "test": "jest"
4   },
5   "jest": {
6     "maxConcurrency": 1,
7     "transform": {
8       "^.+\\.[t|j]sx?$": "babel-jest"
9     },
10    "collectCoverage": true,
11    "collectCoverageFrom": [
12      "./src/**/*.js"
13    ],
14  }
15
```

Todo Function

Todo Function

- Gunakan Todo Function ketika kita berencana membuat unit test, namun dilakukan
- Todo Function akan ditampilkan sebagai summary ketika kita menjalankan unit test, untuk mengingatkan kita
- <https://jestjs.io/docs/api#testtodoname>

Kode : Todo Function



A screenshot of a code editor showing a file named `todo.test.js`. The file contains the following code:

```
1 > test.todo("create test for sumAll() big numbers");
2
3 > test.todo("create test for sum() negative numbers");
4 |
```

The code editor interface includes a tab bar with `JS` and `todo.test.js`, and a status bar at the bottom.

Failing Function

Failing Function

- Dalam membuat unit test, jangan hanya membuat skenario sukses
- Kadang kita juga perlu membuat skenario gagal, atau ekspektasi kita gagal, contoh misal ketika kita mengirim data tidak valid, maka kita berharap kalo kode nya terjadi error
- Pada kasus ini, Jest menyediakan fitur Failing Function
- <https://jestjs.io/docs/api#testfailingname-fn-timeout>

Kode : Say Hello Function



```
sayHello.js
1  export const sayHello = (name) => {
2      if (name) {
3          return `Hello ${name}`;
4      } else {
5          throw new Error("Name is required");
6      }
7  }
```

Kode : Failing Function



A screenshot of a code editor showing a file named `failing.test.js`. The code contains two test cases: one passing test for `sayHello` and one failing test for `sayHello` with a null argument.

```
1 import {sayHello} from "../src/sayHello.js";
2
3 test("sayHello success", () => {
4     expect(sayHello("Eko")).toBe("Hello Eko");
5 })
6
7 test.failing("sayHello error", () => {
8     sayHello(null);
9 });
10
```

Mock

Kesulitan Unit Test

- Saat membuat unit test, kadang kita terkendala dengan bagian kode yang sulit di test
- Misal, terdapat kode yang melakukan interaksi dengan sistem lain, misal database, message broker, third party web service, dan lain-lain
- Jika kita lakukan unit test dengan cara berinteraksi secara langsung, maka akan sulit untuk konsisten, misal unit test pertama sukses, ketika dijalankan lagi, ternyata gagal karena datanya sudah ada yang sebelumnya
- Pada kasus seperti ini, ada baiknya kita melakukan Mocking

Mock

- Dalam pemrograman, Mock Object adalah object tiruan yang kita buat, yang tingkah lakuanya menyerupai dengan object aslinya
- Salah satu fitur dalam Mock Object adalah, kita bisa memberitahu tingkah laku dari object tersebut sesuai dengan yang kita inginkan
- Mock Object ini menjadi sangat cocok untuk kita gunakan ketika melakukan unit test yang berhubungan dengan sistem lain, sehingga kita tidak perlu berinteraksi dengan sistem lain tersebut

Jenis Mock

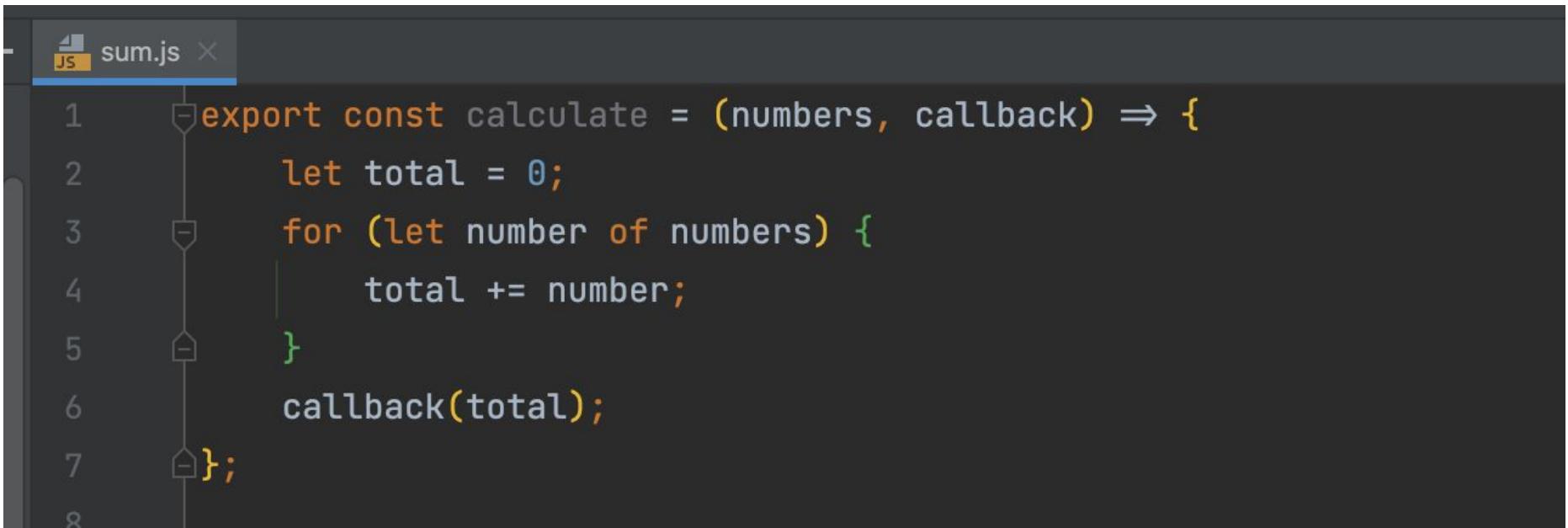
- Jest mendukung banyak jenis Mock, diantaranya
- Mock Function, yang bisa kita gunakan untuk membuat tiruan dari sebuah function
- Mock Class, yang bisa kita gunakan untuk membuat tiruan dari object Class
- Mock Modules, yang bisa kita gunakan untuk membuat tiruan dari Modules

Mock Function

Mock Function

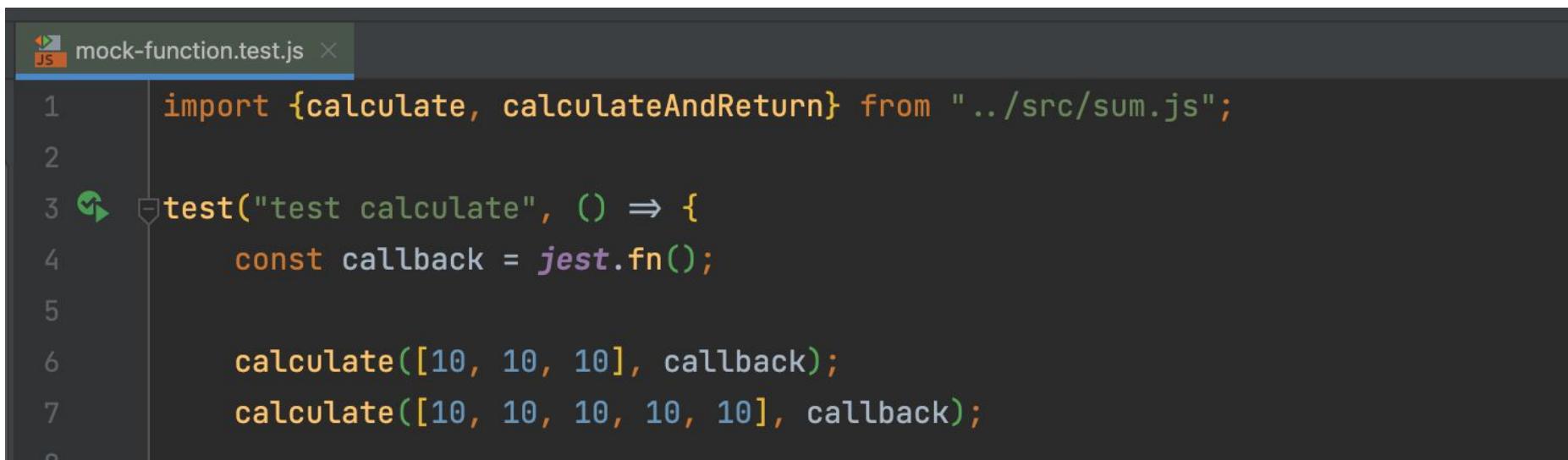
- Jest bisa digunakan untuk membuat mock function
- Dimana kita bisa membuat tiruan dari sebuah function
- Salah satu fitur dari mock function adalah, kita bisa melihat detail dari parameter yang digunakan untuk memanggil mock function tersebut
- Untuk membuat mock function, kita bisa menggunakan `jest.fn()`
- <https://jestjs.io/docs/mock-function-api#jestfnimplementation>

Kode : Calculate Function



```
sum.js
1 export const calculate = (numbers, callback) => {
2     let total = 0;
3     for (let number of numbers) {
4         total += number;
5     }
6     callback(total);
7 };
8
```

Kode : Mock Function



The screenshot shows a code editor window with a dark theme. The title bar says "mock-function.test.js". The code is a Jest test for a "calculate" function:

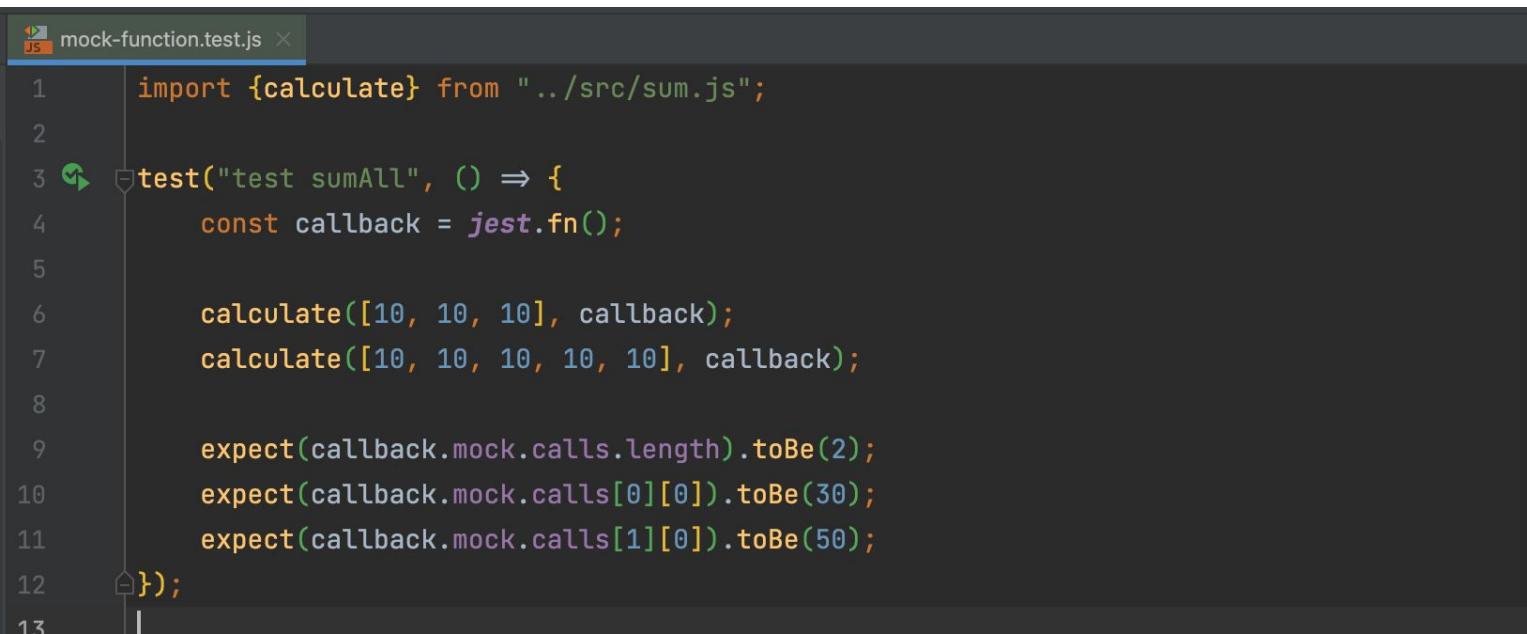
```
1 import {calculate, calculateAndReturn} from "../src/sum.js";
2
3 test("test calculate", () => {
4     const callback = jest.fn();
5
6     calculate([10, 10, 10], callback);
7     calculate([10, 10, 10, 10, 10], callback);
8 }
```

A green circular icon with a white arrow is visible next to the third line of code.

Mock Property

- Saat menggunakan mock, kadang kita ingin melihat parameter apa yang digunakan untuk memanggil mock
- Atau bahkan return value apa yang dikembalikan oleh mock function
- Kita bisa menggunakan property mock dari mock function untuk melihatnya
- <https://jestjs.io/docs/mock-function-api#mockfnmockcalls>
- <https://jestjs.io/docs/mock-function-api#mockfnmockresults>

Kode : Mock Function Property



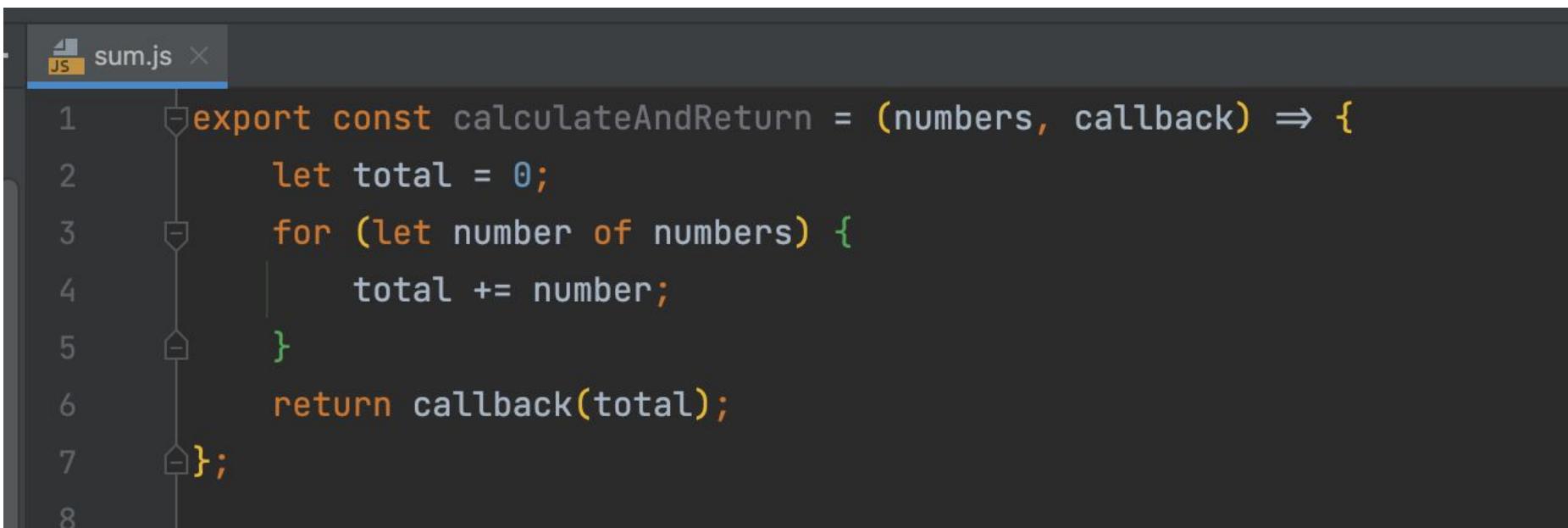
A screenshot of a code editor showing a Jest test file named `mock-function.test.js`. The code uses the `jest.fn()` constructor to create a mock function for the `calculate` function from `src/sum.js`. It then calls `calculate` twice with different arrays of numbers and asserts that the mock function was called twice with the expected arguments.

```
1 import {calculate} from "../src/sum.js";
2
3 test("test sumAll", () => {
4     const callback = jest.fn();
5
6     calculate([10, 10, 10], callback);
7     calculate([10, 10, 10, 10, 10], callback);
8
9     expect(callback.mock.calls.length).toBe(2);
10    expect(callback.mock.calls[0][0]).toBe(30);
11    expect(callback.mock.calls[1][0]).toBe(50);
12 });
13
```

Mock Return Value

- Kadang ada kasus dimana kita ingin membuat mock function dengan return value yang sudah ditentukan
- Namun jika pada kasus sederhana tanpa butuh logic, kita bisa membuat return value dengan mudah pada mock function
- Kita bisa memanfaatkan function `mockReturnValue(returnValue)` atau `mockReturnValueOnce(returnValue)`
- <https://jestjs.io/docs/mock-function-api#mockfnmockreturnvaluevalue>
- <https://jestjs.io/docs/mock-function-api#mockfnmockreturnvalueoncevalue>

Kode : Calculate And Return Function

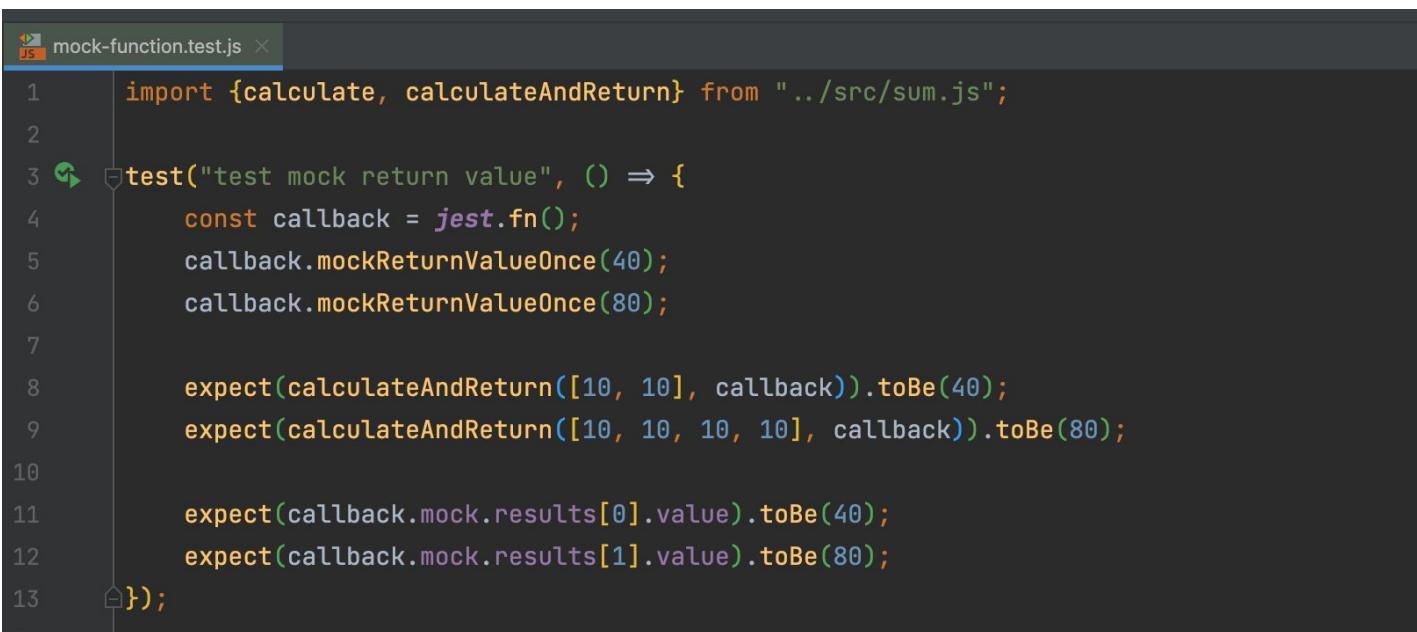


The image shows a screenshot of a code editor with a dark theme. A file named "sum.js" is open, indicated by a tab at the top left. The code editor displays the following JavaScript code:

```
1  export const calculateAndReturn = (numbers, callback) => {
2      let total = 0;
3      for (let number of numbers) {
4          total += number;
5      }
6      return callback(total);
7  };
8
```

The code defines an arrow function named "calculateAndReturn". It takes two arguments: "numbers" and "callback". Inside the function, it initializes a variable "total" to 0, then iterates over the "numbers" array using a for...of loop. For each element, it adds it to "total". Finally, it returns the result of calling the "callback" function with "total" as its argument. The code is numbered from 1 to 8 on the left side.

Kode : Mock Return Value



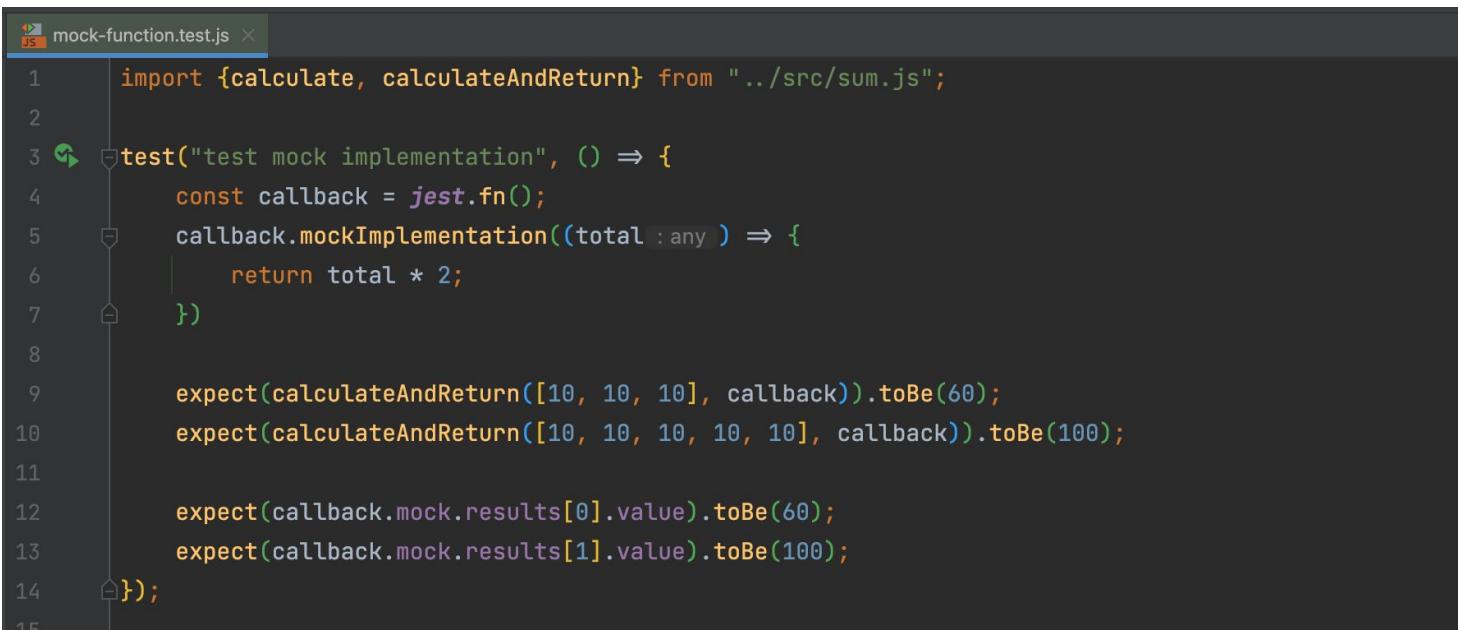
The screenshot shows a code editor window with a dark theme. The tab bar at the top has a file named "mock-function.test.js". The code itself is a Jest test for a function named "calculateAndReturn". It uses the "callback" API provided by Jest to mock the return values of the function. The test cases check the function's behavior with different input arrays and verify the results using Jest's expectation API.

```
1 import {calculate, calculateAndReturn} from "../src/sum.js";
2
3 test("test mock return value", () => {
4     const callback = jest.fn();
5     callback.mockReturnValueOnce(40);
6     callback.mockReturnValueOnce(80);
7
8     expect(calculateAndReturn([10, 10], callback)).toBe(40);
9     expect(calculateAndReturn([10, 10, 10, 10], callback)).toBe(80);
10
11    expect(callback.mock.results[0].value).toBe(40);
12    expect(callback.mock.results[1].value).toBe(80);
13});
```

Mock Implementation

- Pada kasus tertentu, kadang kita ingin membuat implementasi logic dari sebuah mock function
- Dengan menggunakan ini, kita bisa mengubah juga return value dari mock function secara dinamis
- Kita bisa menggunakan `mockImplementation()` atau `mockImplementationOnce()`
- <https://jestjs.io/docs/mock-function-api#mockfnmockimplementationfn>
- <https://jestjs.io/docs/mock-function-api#mockfnmockimplementationoncefn>

Kode : Mock Implementation



A screenshot of a code editor showing a Jest test file named `mock-function.test.js`. The code demonstrates how to mock a function's implementation using `jest.fn()` and its `mockImplementation` method.

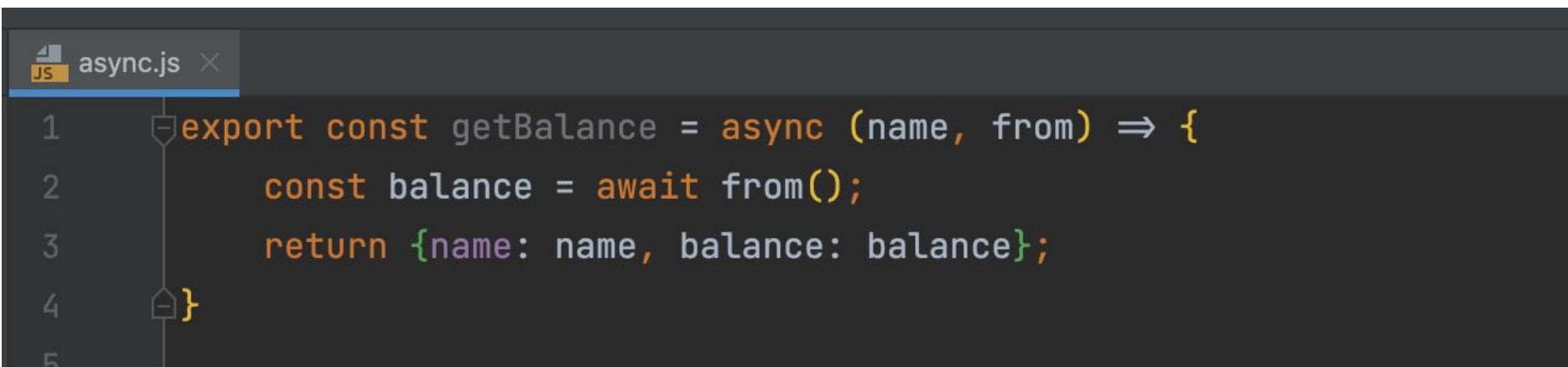
```
1 import {calculate, calculateAndReturn} from "../src/sum.js";
2
3 test("test mock implementation", () => {
4     const callback = jest.fn();
5     callback.mockImplementation((total: any) => {
6         return total * 2;
7     })
8
9     expect(calculateAndReturn([10, 10, 10], callback)).toBe(60);
10    expect(calculateAndReturn([10, 10, 10, 10, 10], callback)).toBe(100);
11
12    expect(callback.mock.results[0].value).toBe(60);
13    expect(callback.mock.results[1].value).toBe(100);
14});
15
```

Mock Async Function

Mock Async Function

- Sebelumnya kita sudah membuat Mock Function
- Jest juga bisa digunakan untuk membuat Mock Function yang bersifat Async, sehingga mengembalikan Promise
- Ada beberapa function yang bisa kita gunakan untuk mengembalikan result Promise
- `mockResolvedValue(value)`
<https://jestjs.io/docs/mock-function-api#mockfnmockresolvedvaluevalue>
- `mockRejectedValue(value)`
<https://jestjs.io/docs/mock-function-api#mockfnmockrejectedvaluevalue>
- Atau kita bisa manual membuat Promise dengan menggunakan `mockImplementation()`

Kode : Get Balance Async



The image shows a screenshot of a code editor with a dark theme. The tab bar at the top has a file named "async.js" selected, indicated by a blue highlight. The code editor displays the following JavaScript code:

```
1  export const getBalance = async (name, from) => {
2      const balance = await from();
3      return {name: name, balance: balance};
4  }
5
```

The code defines an asynchronous function named "getBalance" that takes two parameters: "name" and "from". It uses the await keyword to wait for the result of the "from" function and then returns an object containing "name" and "balance". The code is numbered from 1 to 5 on the left side.

Kode : Get Balance Success

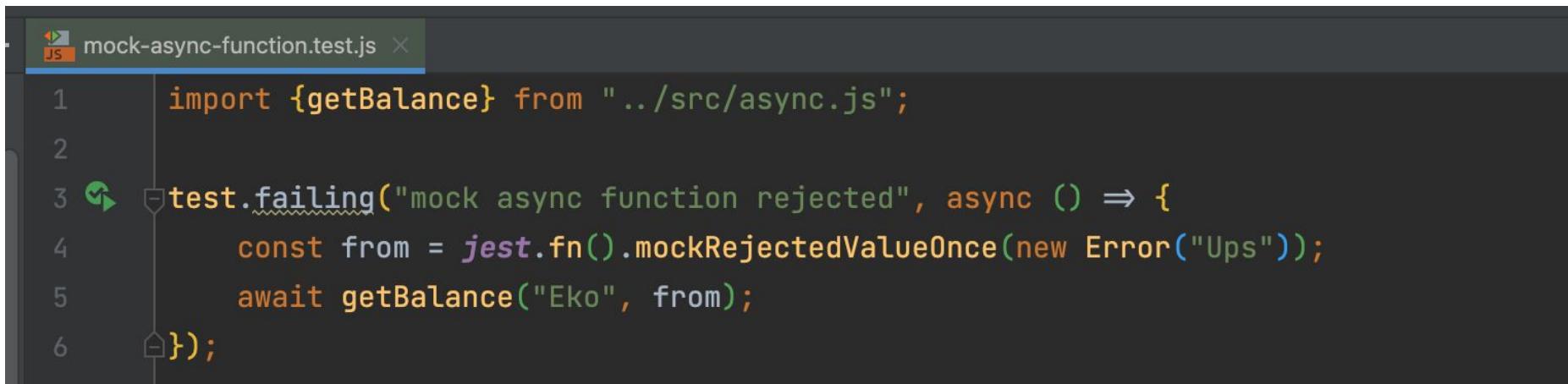


The screenshot shows a code editor window with a dark theme. The file is named `mock-async-function.test.js`. The code is a Jest test for an asynchronous function `getBalance`.

```
1 import {getBalance} from "../src/async.js";
2
3 test("mock async function", async () => {
4     const from = jest.fn().mockResolvedValueOnce(1000);
5     await expect(getBalance("Eko", from)).resolves
6         .toEqual({name: "Eko", balance: 1000});
7
8     await expect(from.mock.calls.length).toBe(1);
9     await expect(from.mock.results[0].value).resolves.toBe(1000);
10});
11
```

The code uses Jest's `fn` and `mockResolvedValueOnce` methods to mock the `getBalance` function. It then uses `expect` to assert that the function was called once with the argument "Eko" and that the result was 1000. Finally, it asserts that the mock's `calls.length` is 1 and that the first result's value is 1000.

Kode : Get Balance Error



The screenshot shows a code editor window with a dark theme. The file tab at the top is labeled "mock-async-function.test.js". The code itself is a Jest test for an asynchronous function named "getBalance". The test uses the "fn" method to mock an asynchronous function and then calls "mockRejectedValueOnce" to trigger a rejection with the message "Ups". The test passes, indicated by a green checkmark icon.

```
1 import {getBalance} from "../src/async.js";
2
3 test.failing("mock async function rejected", async () => {
4     const from = jest.fn().mockRejectedValueOnce(new Error("Ups"));
5     await getBalance("Eko", from);
6 });

```

Mock Matchers

Mock Matchers

- Sebelumnya, untuk mengecek berapa kali mock function dipanggil, atau apa parameter yang diterima oleh mock function, kita lakukan secara manual dengan mengecek data di mock property
- Jest sendiri menyediakan fitur Matchers untuk mock, dimana kita bisa melakukan matchers dengan lebih mudah dibandingkan secara manual

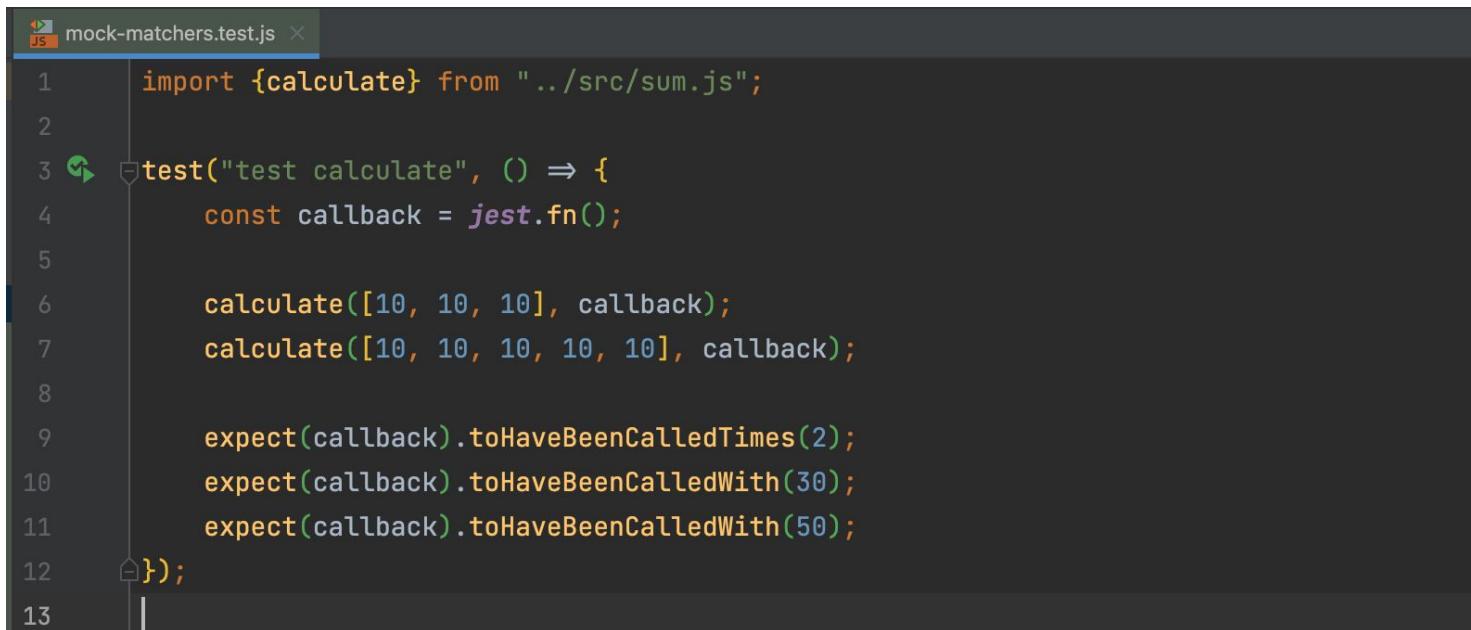


Mock Matchers Function

Function	Keterangan
expect(mock).toHaveBeenCalled()	Memastikan mock pernah dipanggil
expect(mock).toHaveBeenCalledTimes(number)	Memastikan mock pernah dipanggil sebanyak number
expect(mock).toHaveBeenCalledWith(arg1, arg2, ...)	Memastikan mock pernah dipanggil dengan parameter

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

Kode : Mock Matchers



A screenshot of a code editor displaying a Jest test file named `mock-matchers.test.js`. The code uses `jest.fn()` to create a mock function and `expect(callback)` to make assertions about its behavior.

```
1 import {calculate} from "../src/sum.js";
2
3 test("test calculate", () => {
4     const callback = jest.fn();
5
6     calculate([10, 10, 10], callback);
7     calculate([10, 10, 10, 10, 10], callback);
8
9     expect(callback).toHaveBeenCalledTimes(2);
10    expect(callback).toHaveBeenCalledWith(30);
11    expect(callback).toHaveBeenCalledWith(50);
12 });
13
```

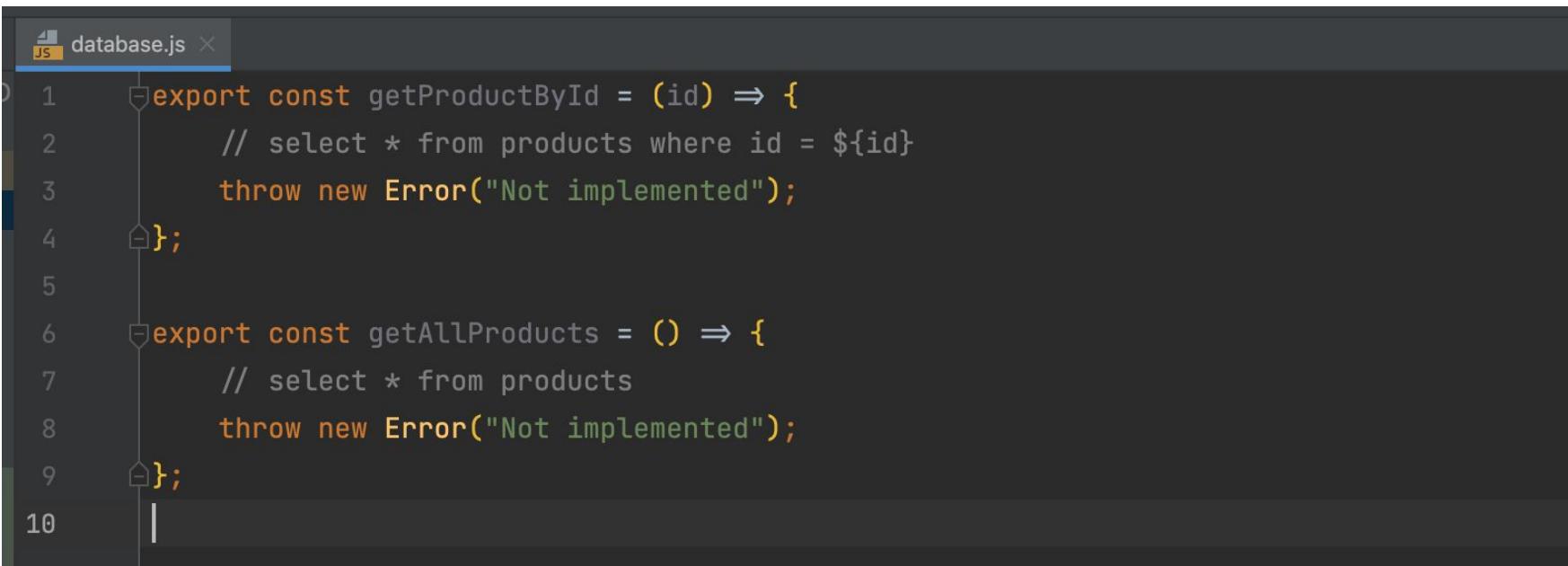
Mock Modules



Mock Modules

- Saat membuat aplikasi, sudah pasti kita akan sering menggunakan JS Module, baik itu yang kita buat sendiri, atau JS Modules opensource
- Jest juga bisa digunakan untuk melakukan mock terhadap modules

Kode : Database Module



The image shows a screenshot of a code editor with a dark theme. The file tab at the top left is labeled "database.js". The code itself is as follows:

```
1  export const getProductById = (id) => {
2      // select * from products where id = ${id}
3      throw new Error("Not implemented");
4  };
5
6  export const getAllProducts = () => {
7      // select * from products
8      throw new Error("Not implemented");
9  };
10 |
```

Kode : Class Product Service



```
product-service.js ×
1 import {getAllProducts, getProductById} from "./database.js";
2
3 export class ProductService {
4
5     static findById(id) {
6         return getProductById(id);
7     }
8
9     static findAll() {
10        return getAllProducts();
11    }
12}
```

Melakukan Mock Module

- Untuk melakukan mock modules, kita bisa gunakan function jest.mock(module)
- Secara otomatis ketika kita import dari module tersebut, maka jest akan melakukan mock
- <https://jestjs.io/docs/jest-object#jestmockmodulename-factory-options>
- Jika kita tidak ingin melakukan mock lagi, kita bisa gunakan jest.unmock()
- <https://jestjs.io/docs/jest-object#jestunmockmodulename>

Kode : Mock Modules (1)



A screenshot of a code editor showing a Jest test file named `mock-modules.test.js`. The code demonstrates how to mock modules in a Jest test. It imports functions from `database.js` and `product-service.js`, then uses `jest.mock` to mock `database.js`. It then defines a test for `getProductById` which uses `getProductById.mockImplementation` to return a specific object. Finally, it calls `findById` on `ProductService` and expects the result to match the mocked object.

```
mock-modules.test.js
1 import {getAllProducts, getProductById} from "../src/database.js";
2 import {ProductService} from "../src/product-service.js";
3
4 jest.mock("../src/database.js");
5
6 test("mock modules getProductById", () => {
7     getProductById.mockImplementation(id => {
8         return {id: id, name: "Product Mock"};
9     })
10
11     const product = ProductService.findById(1);
12     expect(product).toEqual({id: 1, name: "Product Mock"});
13 })
14 }
```

Kode : Mock Modules (2)



A screenshot of a code editor showing a Jest test file named `mock-modules.test.js`. The code demonstrates how to mock modules in a Jest test. It imports `getAllProducts` and `getProductById` from `database.js`, and `ProductService` from `product-service.js`. It then uses `jest.mock` to mock `database.js`. A test suite is defined with a single test case. Inside the test case, it creates a mock product array, sets the `getAllProducts` mock to return this array, and then asserts that `ProductService.findAll` returns the expected products.

```
mock-modules.test.js
1 import {getAllProducts, getProductById} from "../src/database.js";
2 import {ProductService} from "../src/product-service.js";
3
4 jest.mock("../src/database.js");
5
6 test("mock modules getAllProducts", () => {
7     const products = [
8         {id: 1, name: "Product Mock 1"},
9         {id: 2, name: "Product Mock 1"},
10    ];
11    getAllProducts.mockReturnValue(products);
12
13    expect(ProductService.findAll()).toEqual(products);
14})
15
```

Mock Partial Modules

Mock Partial Modules

- Saat kita melakukan mock modules, maka secara default seluruh modules tersebut akan di mock oleh Jest
- Kadang, kita tidak ingin melakukan mock semua bagian di modules, mungkin hanya beberapa bagian saja
- Jest juga memiliki fitur yang bisa kita gunakan untuk menggunakan module aslinya menggunakan `jest.requireActual(module)`
- <https://jestjs.io/docs/jest-object#jestrequireactualmodulename>

Kode : Mock Partial Modules

```
mock-partial-modules.test.js
1 import {getAllProducts} from "../src/database.js";
2 import {ProductService} from "../src/product-service.js";
3
4 jest.mock("../src/database.js", () => {
5   const originalModule = jest.requireActual('../src/database.js');
6
7   return {
8     __esModule: true,
9     ...originalModule,
10    getAllProducts: jest.fn()
11  };
12});
13});
```

Kode : Test Mock Partial Modules

```
test("mock modules getAllProducts", () => {
  const products = [
    {id: 1, name: "Product Mock 1"},
    {id: 2, name: "Product Mock 1"},
  ]
  getAllProducts.mockReturnValue(products);

  expect(ProductService.findAll()).toEqual(products);
}

test.failing("mock modules getProductById", () => {
  ProductService.findById(1);
})
```

Mock Class

Mock Class

- Sebelumnya kita sudah bahas tentang cara melakukan Mock Function, dan juga melakukan mock function di Modules
- Selain Function, Jest juga bisa digunakan untuk melakukan Mock Class
- Seperti yang sudah kita pelajari di JavaScript OOP, Class di JavaScript sebenarnya adalah Constructor Function, yang sebenarnya tidak ada bedanya dengan Function biasanya
- Oleh karena itu, untuk melakukan Mock Class, sama saja seperti kita melakukan Mock Function



Kode : Class UserRepository



```
user-repository.js
1 export class UserRepository {
2
3     save(user) {
4         throw new Error("Not implemented");
5     }
6
7     findById(id) {
8         throw new Error("Not implemented");
9     }
10
11    findAll() {
12        throw new Error("Not implemented");
13    }
14}
```

Kode : Class User Service

```
-service.js ×

import {UserRepository} from "./user-repository.js";

export class UserService {

    constructor(userRepository) {
        if (userRepository) {
            this.userRepository = userRepository;
        } else {
            this.userRepository = new UserRepository();
        }
    }
}
```

```
save(user) {
    this.userRepository.save(user);
}

findById(id) {
    return this.userRepository.findById(id);
}

findAll() {
    return this.userRepository.findAll();
}
```

Kode : Mock Class (1)



The screenshot shows a code editor window with a dark theme. The file being edited is named "mock-class.test.js". The code is a Jest test for a User Service, which depends on a UserRepository. The test uses a mock repository and asserts that the save method is called on the repository.

```
mock-class.test.js
1 import {UserRepository} from "../src/user-repository.js";
2 import {UserService} from "../src/user-service.js";
3
4 jest.mock('../src/user-repository.js');
5
6 const repository = new UserRepository();
7 const service = new UserService(repository);
8
9 test('test mock class save', () => {
10   const user = {id: 1, name: "Eko"}
11   service.save(user);
12   expect(repository.save).toHaveBeenCalled();
13 });
14
```

Kode : Mock Class (2)

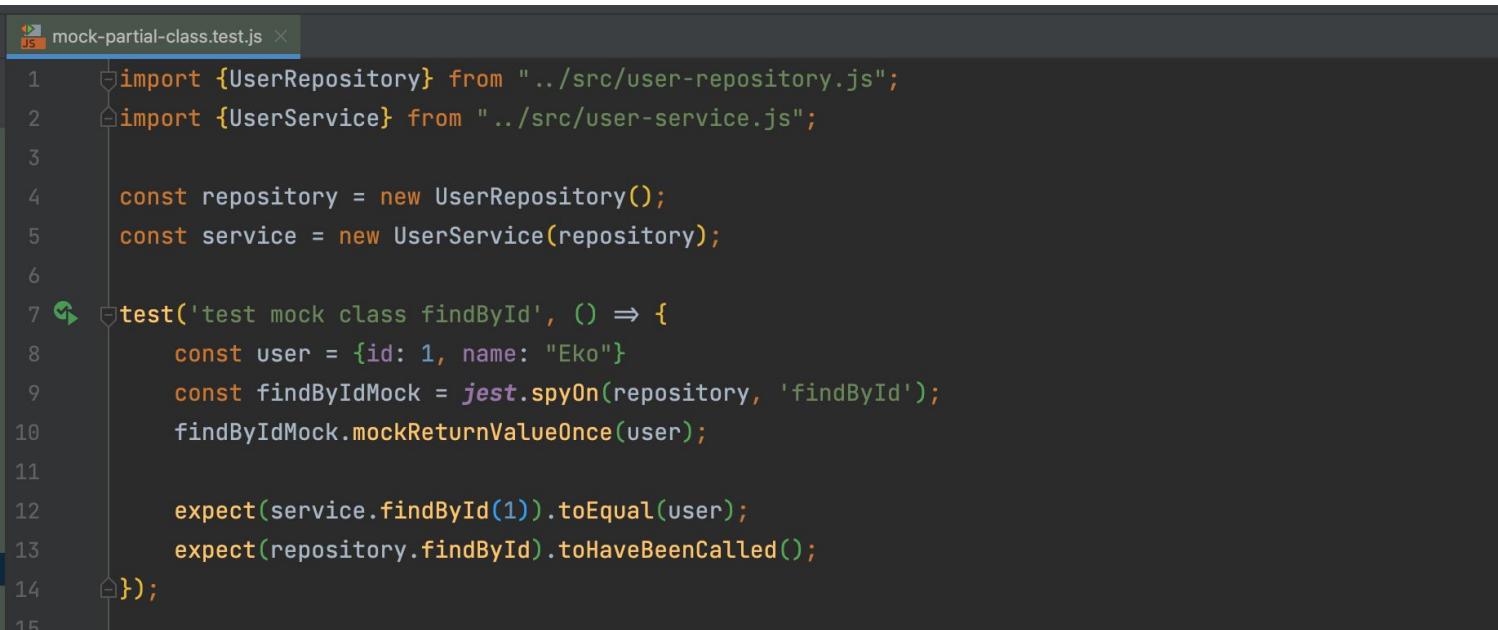
```
15  ↵  test('test mock class findById', () => {
16      const user = {id: 1, name: "Eko"}
17      repository.findById.mockReturnValueOnce(user);
18      expect(service.findById(1)).toEqual(user);
19      expect(repository.findById).toHaveBeenCalled();
20  });
21
22  ↵  test('test mock class findAll', () => {
23      const users = [{id: 1, name: "Eko"}, {id: 2, name: "Kurniawan"}];
24      repository.findAll.mockReturnValueOnce(users);
25      expect(service.findAll()).toEqual(users);
26      expect(repository.findAll).toHaveBeenCalled();
27  })
28
```

Mock Partial Class

Mock Partial Class

- Saat kita melakukan mock class dengan melakukan mock modules, secara otomatis semua function di class tersebut akan ter-mock
- Kadang, misal ada kasus dimana kita ingin melakukan mock hanya sebagian function saja di dalam class
- Pada kasus ini, kita bisa menggunakan jest.spyOn()
- <https://jestjs.io/docs/jest-object#jestspyonobject-methodname>

Kode : Mock Partial Class



The screenshot shows a code editor window with a dark theme. The file is named `mock-partial-class.test.js`. The code is a Jest test for a `UserService` class that depends on a `UserRepository`.

```
mock-partial-class.test.js
1 import {UserRepository} from "../src/user-repository.js";
2 import {UserService} from "../src/user-service.js";
3
4 const repository = new UserRepository();
5 const service = new UserService(repository);
6
7 test('test mock class findById', () => {
8     const user = {id: 1, name: "Eko"};
9     const findByIdMock = jest.spyOn(repository, 'findById');
10    findByIdMock.mockReturnValueOnce(user);
11
12    expect(service.findById(1)).toEqual(user);
13    expect(repository.findById).toHaveBeenCalled();
14});
```

Materi Selanjutnya

Materi Selanjutnya

- ExpressJS
- NodeJS Database
- Dan lain-lain