Get Started

Advanced JavaScript

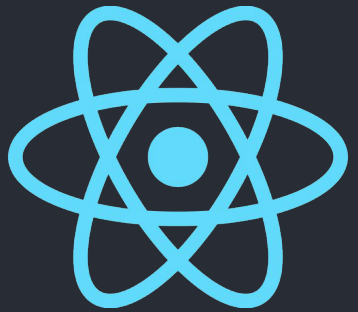


Table of Contents

Destructing

Destructing

- Mengekstrak (unpack) array atau object ke variable.
- Destructing object berdasarkan key.
- Destructing array berdasarkan urutan(index).

Referensi: JavaScript Info - Destructing.

destructing-object.js

```
// Membuat object
const req = {
  body: {
    name: "Aufa",
    age: 22,
    major: "Informatics",
  },
};

// Memanggil nilai name, age, dan major
console.log(req.body.name, req.body.age, req.body.major);
```

destructing-object.js

```
// Membuat object
const req = {
  body: {
    name: "Aufa",
    age: 22,
    major: "Informatics",
  },
};

// Menyimpan nilai object ke variable terpisah
const name = req.body.name;
const age = req.body.age;
const major = req.body.major;

console.log(name, age, major);
```

destructing-object.js

```
// Membuat object
const req = {
  body: {
    name: "Aufa",
    age: 22,
    major: "Informatics",
  },
};

/**
 * Melakukan destructing object.
 * Destructing object berdasarkan key.
 */
const { name, age, major } = req.body;

console.log(name, age, major);
```

destructing-array.js

```js
/**
 * Membuat array family. Terdiri dari:
 * - Suami: Michael
 * - Istri: Hannah
 * - Anak: Jonas
 */
const family = ["Michael", "Hannah", "Jonas"];

// Menampilkan nilai array.
console.log(family[0], family[1], family[2]);
```

destructing-array.js

```js
/**
 * Membuat array family. Terdiri dari:
 * - Suami: Michael
 * - Istri: Hannah
 * - Anak: Jonas
 */
const family = ["Michael", "Hannah", "Jonas"];

// Menyimpan nilai array ke variable terpisah
const husband = family[0];
const wife = family[1];
const son = family[2];

// Menampilkan nilai array.
console.log(husband, wife, son);
```

destructing-array.js

```
/**
 * Membuat array family. Terdiri dari:
 * - Suami: Michael
 * - Istri: Hannah
 * - Anak: Jonas
 */
const family = ["Michael", "Hannah", "Jonas"];

/**
 * Melakukan Array Destructing.
 * Array Destructing berdasarkan posisi atau index.
 */
const [husband, wife, son] = family;

// Menampilkan nilai array.
console.log(husband, wife, son);
```

Rest and Spread

Rest Parameters

Rest parameters dan spread syntax menggunakan keyword titik tiga (...).

Rest parameters:

- Menggabungkan items/parameters menjadi satu (array).

Referensi: JavaScript Info - Rest and Spread.

without rest parameters

```
/**
 * Membuat fungsi sum.
 * Menjumlahkan semua bilangan.
 * Fungsi memiliki 2 parameter
 */

function sum(a, b) {
  const hasil = a + b;
  return hasil;
}

sum(1, 2);
```

without rest parameters

```
/**
 * Membuat fungsi sum.
 * Menjumlahkan semua bilangan.
 * Fungsi memiliki 5 parameter
 */

function sum(a, b, c, d, e) {
  const hasil = a + b + c + d + e;
  return hasil;
}

sum(1, 2, 3, 4, 5);
```

rest-parameters.js

```js
/**
 * Membuat fungsi sum.
 * Menjumlahkan semua bilangan.
 * Menggunakan rest parameter.
 */
function sum(...numbers) {
  let hasil = 0;

  for (const number of numbers) {
    hasil += number;
  }

  return hasil;
}

console.log(sum(1, 2, 3, 4, 5));
```

rest-parameters.js

```js
/**
 * Membuat fungsi showFamilies.
 * Menampilkan anggota keluarga.
 * Parameter ketiga menggunakan rest parameters.
 */

function showFamilies(husband, wife, ...children) {
  console.log(`Suami: ${husband}`);
  console.log(`Istri: ${wife}`);

  for (const child of children) {
    console.log(`Anak: ${child}`);
  }
}

showFamilies("Mikel", "Hannah", "Jonas", "Martha", "Magnuz");
```

Spread Syntax

Rest parameters dan spread syntax menggunakan keyword titik tiga (...).

Spread syntax:

- Menyebarakan/memisahkan items (array atau object) menjadi item tersendiri.
- Spread dapat digunakan pada array atau object.

Tujuan:

- Copy array atau object.
- Merge array atau object.

Referensi: JavaScript Info - Rest and Spread.

without spread

```
/**
 * Copy families ke newFamilies.
 */
const families = ["Mikel", "Hannah"];

const newFamilies = [families[0], families[1], "Jonas", "Martha"];

console.log(newFamilies);
```

spread.js

```
/**
 * Spread operator: Memisahkan array menjadi nilai tersendiri.
 * Kegunaan: Copy dan merge array ke variable baru.
 */
const families = ["Mikel", "Hannah"];

const newFamilies = [...families, "Jonas", "Martha"];
```

without spread

```
/**
 * Copy object user ke newUser
 */
const user = {
  name: "Aufa",
  major: "Informatics",
};

const newUser = {
  name: user.name,
  major: user.major,
  age: 22,
};
```

spread.js

```
/**
 * Spread operator: Memisahkan object menjadi nilai tersendiri.
 * Kegunaan: Copy dan Merge object ke variable baru.
 */

const user = {
  name: "Aufa",
  major: "Informatics",
};

const newUser = {
  ...user,
  age: 22,
};
```

Array Method

Array Method

Array menyediakan banyak method untuk mempermudah mengelola data.

Methods:

- Menambah dan menghapus: push, pop, shift, unshift.
- Iterate (perulangan): forEach.
- Searching (mencari): find, filter.
- Transform (mengubah): map, reduce.

Referensi: JavaScript Info - Array Methods.

foreach.js

```
// Membuat array names
const names = ["Michael", "Hannah", "Jonas"];

/**
 * Menjalankan method forEach.
 * Method forEach bertujuan untuk iterasi.
 */
names.forEach(function (name) {
  console.log(`Nama: ${name}`);
});
```

map.js

```
// Membuat array names
const names = ["Michael", "Hannah", "Jonas"];

/**
 * Menjalankan method map.
 * Method map untuk transform/mengubah data.
 */
const formattedName = names.map(function (name) {
  return `Mr/Mrs. ${name}`;
});

console.log(formattedName);
```

find.js

```
// Membuat array of object
const users = [
  {
    name: "Jonas",
    age: 15,
  },
  {
    name: "Michael",
    age: 40,
  },
  {
    name: "Hannah",
    age: 35,
  },
];

/**
 * Menjalankan method find.
 * Mencari 1 data berdasarkan kondisi tertentu.
 * Kondisi: umur lebih dari 21
 */
const foundUser = users.find(function (user) {
  return user.age > 21;
});

console.log(foundUser);
```

filter.js

```
// Membuat array of object
const users = [
  {
    name: "Jonas",
    age: 15,
  },
  {
    name: "Michael",
    age: 40,
  },
  {
    name: "Hannah",
    age: 35,
  },
];

/**
 * Menjalankan method filter.
 * Mencari semua data berdasarkan kondisi tertentu.
 * Kondisi: umur lebih dari 21
 */
const filteredUser = users.filter(function (user) {
  return user.age > 21;
});

console.log(filteredUser);
```

array methods with array function

```
// forEach dengan arrow function
names.forEach((name) => console.log(`Nama: ${name}`));

// map dengan arrow function
const formattedName = names.map((name) => `Mr/Mrs. ${name}`);

// find dengan arrow function
const foundUser = users.find((user) => user.age > 21);

// filter dengan arrow functio
const filteredUser = users.filter((user) => user.age > 21);
```

Modules

Modules

- Modules adalah sebuah file yang berisi kode.
- Kode dipisahkan berdasarkan tujuan atau responsibility tertentu (SRP).
- Bertujuan: mengurangi kompleksitas, memudahkan maintenance, penerapan arsitektur.
- Modules adalah prinsip penting dalam penerapan berbagai pattern.

Jenis:

- ES Module: Berjalan di browser.
- CommonJS: Berjalan di luar browser (NodeJS).

Referensi: JavaScript Info - Modules.

app.js

```
// Membuat array users. Berisi data users.
const users = [
  { name: "Jonas", age: 15 },
  { name: "Michael", age: 40 },
  { name: "Hannah", age: 35 },
];

/**
 * Membuat Controller User.
 * Terdapat method index dan store.
 */
const index = () => {
  users.forEach(function (user) {
    console.log(user);
  });
};

const store = (user) => {
  users.push(user);
};

/**
 * Membuat fungsi utama.
 * Fungsi ini yang dijalankan pertama kali.
 */
const main = () => {
  const user = { name: "Martha", age: 20 };

  index();
  store(user);
};

main();
```

Dari pada menuliskan semua kode dalam 1 file, kita dapat memisahkan kode berdasarkan tujuan atau responsibility:

- users dapat disimpan di file data.js
- method index dan store dapat disimpan di file controller.js
- method main dapat disimpan di file app.js

Dengan memisahkan kode (modules) berdasarkan tujuan, maka aplikasi menjadi lebih terstruktur (arsitektur).

data.mjs

```mjs
// Membuat array of object
const users = [
  { name: "Jonas", age: 15 },
  { name: "Michael", age: 40 },
  { name: "Hannah", age: 35 },
];

/**
 * Export 1 buah data: users.
 * Export menggunakan keyword default.
 */
export default users;
```

controller.mjs

```mjs
// Import data users dari file data.mjs
import users from "./data.mjs";

/**
 * Membuat User Controller.
 * Terdapat method index dan store.
 */
const index = () => {
  console.log("Index - Get All Users");
  users.forEach(function (user) {
    console.log(user);
  });
};
const store = (user) => {
  users.push(user);
};

/**
 * Export beberapa data.
 * Data disimpan di dalam object.
 */
export { index, store };
```

controller.mjs

```mjs
/**
 * Import User Controller dari file controller.mjs
 * Melakukan destructing untuk extrak object hasil import.
 */
import { index, store } from "./controller.mjs";

const main = () => {
  const user = { name: "Martha", age: 22 };

  index();
  store(user);
  index();
}

main();
```

Promises

Asynchronous

- Proses yang berjalan tanpa harus menunggu proses lain selesai.
- Operasi Asynchronous:
  - Mengakses database.
  - Mengakses file.
  - Mengakses jaringan (resource, fetch)
- JavaScript memiliki fungsi setTimeOut yang menjalankan kode secara asynchronous.
- setTimeOut: menjalankan kode setelah waktu tertentu.

Referensi: JavaScript Info - Callbacks.

set-time-out.js

```
function download() {
  setTimeout(() => {
    console.log("Downloading...");
  }, 3000);
}

function verify() {
  setTimeout(() => {
    console.log("Verify file...");
  }, 2000);
}

function notify() {
  console.log("Download complete");
}

function main() {
  download();
  verify();
  notify();
}

main();
```

- Hasil yang muncul tidak sesuai.
- Karena operasi asynchronous tidak mencegah (non-blocking) proses selanjutnya.
- Solusi: callback.

set-time-out.js

```
function download(callVerify) {
  setTimeout(() => {
    console.log("Downloading...");

    callVerify();
  }, 3000);
}

function verify(callNotify) {
  setTimeout(() => {
    console.log("Verify file...");

    callNotify();
  }, 2000);
}

function notify() {
  console.log("Download complete");
}
```

set-time-out.js

```
const main = () => {
  download(function () {
    verify(function () {
      notify();
    });
  });
};

main();
```

callback-hell.js

```
function main() {
  download(function () {
    verify(function () {
      clearCache(function () {
        callbackAgain(function () {
          callbackAgain(function () {
            notify();
          });
        });
      });
    });
  });
}

main();
```

- Bagaimana jika ada 10 operasi asynchronous?
- Problem: Callback Hell.
- Solusi: Promise.

Promises

- Object yang mengembalikan nilai di masa mendatang (future).
- Solusi untuk menghandle proses Asynchronous.

Pembuatan promise:

1. Producing: Membuat Promise.
2. Consuming: Menggunakan Promise.

Promise memiliki 3 keadaan (state):

- Pending: Ketika promise dijalankan.
- Fulfilled: Ketika promise berhasil (resolve).
- Rejected: Ketika promise gagal (reject).

Referensi: JavaScript Info - Promises.

promise.js

```
function download() {
  /**
   * Promise dibuat menggunakan class Promise.
   * Promise menerima callback/executor.
   * Executor menerima 2 params: resolve, reject.
   * resolve untuk mengembalikan promise berhasil.
   * reject untuk mengembalikan promise gagal.
   */
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("Downloading...");
    }, 3000);
  });
}

function verify() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("Verify file...");
    }, 2000);
  });
}

function notify() {}
  console.log("Download complete");
}
```

Promises - Consuming

Tahapan sebelumnya merupakan tahapan Producing.

Selanjutnya kita perlu melakukan Consuming, yaitu menggunakan hasil promise.

Consuming Promise:

- .then: menangkap promise ketika status berhasil (resolve).
- .catch: menangkap promise ketika status gagal (reject).

Referensi: JavaScript Info - Consuming.

promise.js

```
function download() {
  /**
   * Promise dibuat menggunakan class Promise.
   * Promise menerima callback/executor.
   * Executor menerima 2 params: resolve, reject.
   * resolve untuk mengembalikan promise berhasil.
   * reject untuk mengembalikan promise gagal.
   */
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("Downloading...");
    }, 3000);
  });
}

function verify() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("Verify file...");
    }, 2000);
  });
}

function notify() {}
  console.log("Download complete");
}
```

promise.js

```
download()
  .then((results) => {
    console.log(results);
    return verify();
  })
  .then((results) => {
    console.log(results);
    notify();
  })
  .catch((error) => {
    console.log(error);
  });
```

Fetch

- Fetch merupakan fungsi yang dapat digunakan untuk mengakses API (AJAX).
- Fetch dibangun di atas Promise.
- Untuk menggunakan Fetch perlu memahami promise.

Referensi: JavaScript Info - Fetch.

fetch.js

```
const download = () => {
  /**
   * Menjalankan fungsi fetch.
   * Fetch menerima parameter url dari API.
   * Fetch dibangun di atas promise.
   */
  fetch("https://jsonplaceholder.typicode.com/users")
    .then((results) => {
      return results.json();
    })
    .then((results) => {
      console.log(results);
    })
    .catch((error) => {
      console.log(error);
    });
};

download();
```

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Document</title>
  </head>
  <body>
    <script src="fetch.js"></script>
  </body>
</html>
```

Async Await

- Async Await: menulis kode asynchronous dengan gaya synchronous.
- Async Await dibangun di atas Promise.
- Async Await hanya berjalan di dalam function.

Referensi: JavaScript Info - Async Await.

async-await.js

```
function download() {
  /**
   * Promise dibuat menggunakan class Promise.
   * Promise menerima callback/executor.
   * Executor menerima 2 params: resolve, reject.
   * resolve untuk mengembalikan promise berhasil.
   * reject untuk mengembalikan promise gagal.
   */
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("Downloading...");
    }, 3000);
  });
}

function verify() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("Verify file...");
    }, 2000);
  });
}

function notify() {}
  console.log("Download complete");
}
```

async-await.js

```
/**
 * Membuat async await dengan arrow function.
 * async: memberitahu ada proses asynchronous.
 * await: menunggu proses asynchronous selesai.
 */
const main = async () => {
  console.log(await download());
  console.log(await verify());
  notify();
};

main();
```

callback

```
const main = () => {
  download(function () {
    verify(function () {
      notify();
    });
  });
};

main();
```

promise

```
const main = () => {
  download()
    .then((results) => {
      console.log(results);
      return verify();
    })
    .then((results) => {
      console.log(results);
      notify();
    })
    .catch((error) => {
      console.log(error);
    });
};

main();
```

async await

```
const main = async () => {
  console.log(await download());
  console.log(await verify());
  notify();
};

main();
```

Which is better and readable?

QnA

Take Away

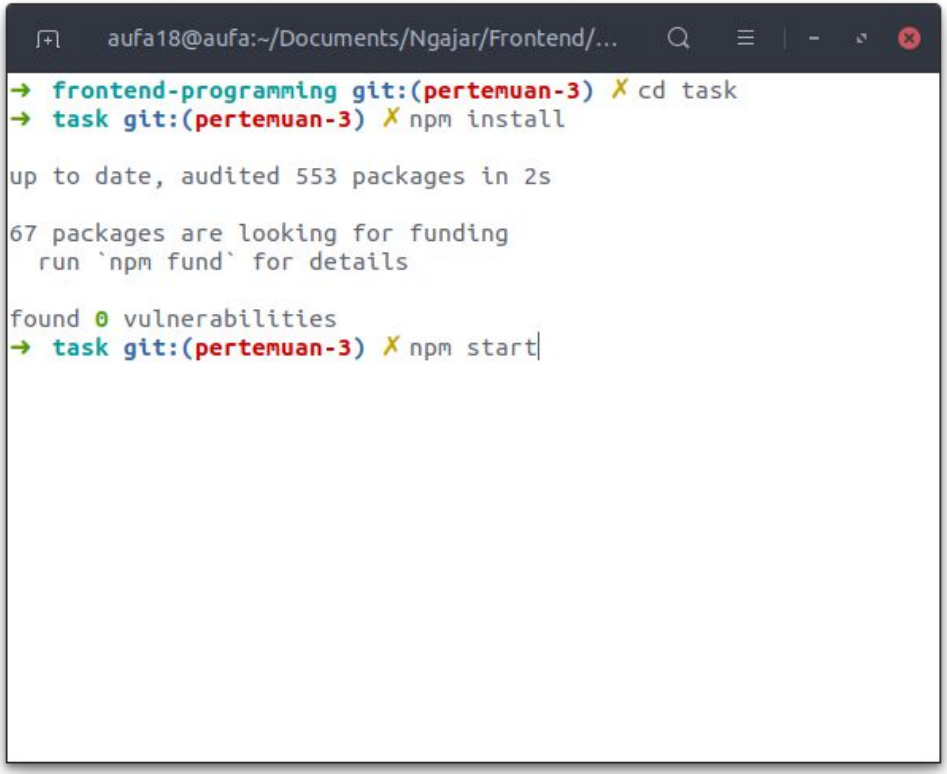
Task

Description:

- Task boilerplate and description: Link.
- Practice Modules and Promise.
- Practice Modern JavaScript before learning React.
- Practice TDD (Test Driven Development).

Assignment:

- Make repo private. Invite: aufaroot18, ekaameliaa, alfkri.
- Push code to Repository Github (use branch for task management).
- Task tidak perlu dizip dan folder node_nodules tidak perlu diupload.
- Submit link repository to elena: Link.



Menjalankan task:

- Masuk ke folder task: cd task
- Install dependencies: npm install
- Jalankan task: npm start

Note:

- Kodingan task di folder src.
- File lain tidak perlu diubah

Menjalankan test:

- Jalankan test: npm test
- Test harus berhasil semua
- Folder test tidak perlu diubah

users.js

```
const users = [
  { name: "Aufa", age: 22, major: "Frontend Engineer" },
  { name: "Isfa", age: 20, major: "Android Engineer" },
  { name: "Agung", age: 24, major: "Data Engineer" },
  { name: "Nurul", age: 24, major: "English" },
  { name: "Jaki", age: 27, major: "English" },
];

/**
 * TODO 1.
 * Export variable users.
 * Gunakan export default.
 */
// CODE HERE
```

UserController.js

```
/**
 * TODO 2.
 * Import variable users dari file data/users.js
 */
// CODE HERE
```

UserController.js

```
/**
 * SARAN TODO3 - TODO5.
 * Tulis dulu solusi tanpa penggunaan promise.
 * Setelah itu refactor dengan menambahkan promise.
 */

/**
 * TODO 3.
 * Buat function formatUser: Format nama user.
 * Fungsi membutuhkan waktu 3 detik.
 * Hint:
 * - Gunakan method map untuk format user.
 * - Gunakan promise untuk handle asynchronous.
 */
const formatUser = (title) => {};
```

app.js

```
(async () => {
  console.log("# Format User: Mr/Mrs");
  const resultFormatUsers = await formatUser("Mr/Mrs");
  console.log(resultFormatUsers);

  console.log("\n# Find User by Name: Aufa");
  const resultFindByName = await findByName("Aufa");
  console.log(resultFindByName);

  console.log("\n# Filter User by Major: English");
  const resultFilterByMajor = await filterByMajor("English");
  console.log(resultFilterByMajor);
})();
```

forrmatUser - Format nama user

```
[
  { name: 'Mr/Mrs. Aufa', age: 22, major: 'Frontend Engineer' },
  { name: 'Mr/Mrs. Isfa', age: 20, major: 'Android Engineer' },
  { name: 'Mr/Mrs. Agung', age: 24, major: 'Data Engineer' },
  { name: 'Mr/Mrs. Nurul', age: 24, major: 'English' },
  { name: 'Mr/Mrs. Jaki', age: 27, major: 'English' }
]
```

UserController.js

```
/**
 * SARAN TODO3 - TODO5.
 * Tulis dulu solusi tanpa penggunaan promise.
 * Setelah itu refactor dengan menambahkan promise.
 */

/**
 * TODO 4.
 * Buat function findByName: Mencari 1 user by name.
 * Fungsi membutuhkan waktu 2 detik.
 * Hint:
 * - Gunakan method find untuk mencari 1 user.
 * - Gunakan promise untuk handle asynchronous.
 */
const findByName = (name) => {};
```

app.js

```
(async () => {
  console.log("# Format User: Mr/Mrs");
  const resultFormatUsers = await formatUser("Mr/Mrs");
  console.log(resultFormatUsers);

  console.log("\n# Find User by Name: Aufa");
  const resultFindByName = await findByName("Aufa");
  console.log(resultFindByName);

  console.log("\n# Filter User by Major: English");
  const resultFilterByMajor = await filterByMajor("English");
  console.log(resultFilterByMajor);
})();
```

findByName - Find 1 user by name

```
{ name: 'Aufa', age: 22, major: 'Frontend Engineer' }
```

UserController.js

```
/**
 * SARAN TODO3 - TODO5.
 * Tulis dulu solusi tanpa penggunaan promise.
 * Setelah itu refactor dengan menambahkan promise.
 */

/**
 * TODO 5.
 * Buat function filterByMajor: Mencari semua user by major.
 * Fungsi membutuhkan waktu 4 detik.
 * Hint:
 * - Gunakan method filter untuk mencari semua user.
 * - Gunakan promise untuk handle asynchronous.
 */
const filterByMajor = (major) => {};
```

app.js

```
(async () => {
  console.log("# Format User: Mr/Mrs");
  const resultFormatUsers = await formatUser("Mr/Mrs");
  console.log(resultFormatUsers);

  console.log("\n# Find User by Name: Aufa");
  const resultFindByName = await findByName("Aufa");
  console.log(resultFindByName);

  console.log("\n# Filter User by Major: English");
  const resultFilterByMajor = await filterByMajor("English");
  console.log(resultFilterByMajor);
})();
```

filterByMajor - Filter all users by major

```
[
  { name: 'Nurul', age: 24, major: 'English' },
  { name: 'Jaki', age: 27, major: 'English' }
]
```

UserController.js

```
/**
 * TODO 6.
 * Export fungsi: formatUser, findByName, filterByMajor
 */
// CODE HERE
```

app.js

```
/**
 * TODO 6.
 * Import fungsi formatUser, findByName, fildetByMajor
 * dari file controllers/UserController.js
 */
// CODE HERE

/**
 * Fungsi Main.
 * Jangan edit atau hapus fungsi main.
 * Fungsi main untuk testing aplikasi.
 */
(async () => {
  console.log("# Format User: Mr/Mrs");
  const resultFormatUsers = await formatUser("Mr/Mrs");
  console.log(resultFormatUsers);

  console.log("\n# Find User by Name: Aufa");
  const resultFindByName = await findByName("Aufa");
  console.log(resultFindByName);

  console.log("\n# Filter User by Major: English");
  const resultFilterByMajor = await filterByMajor("English");
  console.log(resultFilterByMajor);
})();
```

```
PASS test/task.test.js
  # Menjalankan Test: Array of Object Users
    ✓ users harus berupa array of object (3 ms)
    ✓ users setidaknya memiliki 5 items/data
    ✓ Aufa harus berupa object
    ✓ Isfhani Ghiyath harus berupa object
    ✓ Fai harus berupa object (1 ms)
    ✓ Agung harus berupa object (11 ms)
    ✓ Sabiq harus berupa object (1 ms)
    ✓ Aufa harus memiliki property name, age, dan major (2 ms)
    ✓ Isfhani Ghiyath harus memiliki property name, age, dan major (1 ms)
    ✓ Fai harus memiliki property name, age, dan major (1 ms)
    ✓ Agung harus memiliki property name, age, dan major (1 ms)
    ✓ Sabiq harus memiliki property name, age, dan major (1 ms)
  # Menjalankan Test: Function all
    ✓ function all harus bisa dipanggil (7 ms)
  # Menjalankan Test: Function store
    ✓ function create harus bisa dipanggil (7 ms)
    ✓ function create harus bisa menambahkan user baru (10 ms)
  # Menjalankan Test: Function edit
    ✓ function edit harus bisa dipanggil (8 ms)
    ✓ function edit harus bisa mengedit user (9 ms)
  # Menjalankan Test: Function destroy
    ✓ function destroy harus bisa dipanggil (8 ms)
    ✓ function destroy harus bisa menghapus user (7 ms)

Test Suites: 1 passed, 1 total
Tests:       19 passed, 19 total
Snapshots:   0 total
Time:        0.466 s, estimated 1 s
Ran all test suites.
```

Menjalankan test:

- Jalankan test: npm test
- Test harus berhasil semua.
- Folder test tidak perlu diubah

Attendance

Thanks