**Skenario Update Note**

Selanjutnya kita tulis pengujian untuk skenario ke-4, yakni memperbaharui data catatan. Seperti biasa, kita ingat-ingat dulu kriteria pengujiannya yah.

Skenario 4: Update Note (Memperbarui data catatan)

Pastikan response memiliki status code 200.

Pastikan header response Content-Type memiliki nilai application/json.

Pastikan body response adalah object.

Pastikan body response memiliki properti dan nilai yang sesuai.

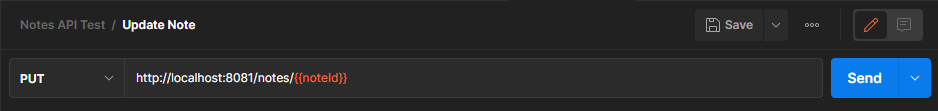
Ketika mengakses catatan yang diperbaharui

Pastikan catatan yang diperbarui memiliki nilai terbaru.

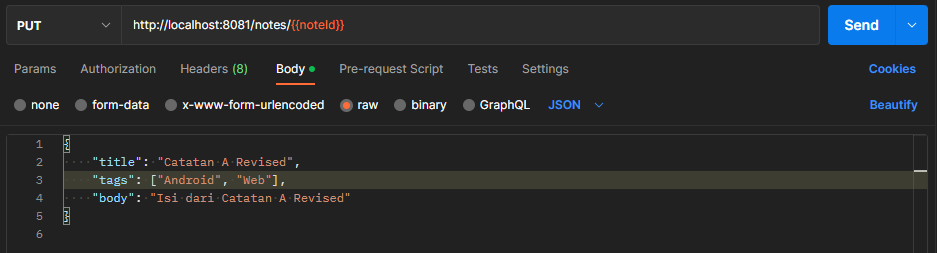
Sudah mantap? Yuk kita mulai tuliskan pengujiannya.

Buat request baru di dalam collection dengan nama “Update Note”.

Pada request URL, isikan nilai localhost:5000/notes/{{noteId}} dan ubah method yang digunakan menjadi PUT.



Sebelum menuliskan tes, sisipkan dulu data catatan terbaru pada body request. Silakan pilih tab Body, kemudian pilih raw, dan ubah tipenya menjadi JSON.



Kemudian tulis JSON berikut pada body request.

{

"title": "Catatan A Revised",

"tags": ["Android", "Web"],

"body": "Isi dari Catatan A Revised"

}

Data sudah siap! Lanjut pilih tab Tests untuk mulai menuliskan pengujian.

Seperti biasa, pengujian pertama kita uji status code dari respons. Silakan buat test dengan nama “response status code should have 200 value”.

pm.test('response status code should have 200 value', () => {

});

Di dalamnya, uji response code dengan nilai 200.

pm.test('response status code should have 200 value', () => {

pm.response.to.have.status(200);

});

Done!

Skenario 4: Update Note (Memperbarui data catatan)

~~Pastikan response memiliki status code 200.~~

Pastikan header response Content-Type memiliki nilai application/json.

Pastikan body response adalah object.

Pastikan body response memiliki properti dan nilai yang sesuai.

Ketika mengakses catatan yang diperbaharui

Pastikan catatan yang diperbarui memiliki nilai terbaru.

Lanjut kita uji nilai Content-Type yang ada pada response. Silakan buat test baru dengan nama “‘response Content-Type header should have application/json value”.

pm.test('response Content-Type header should have application/json value', () => {

});

Seperti biasa, di dalamnya kita uji response header Content-Type dengan nilai “application/json; charset=utf-8”.

pm.test('response Content-Type header should have application/json value', () => {

pm.expect(pm.response.headers.get('Content-Type')).to.equals("application/json; charset=utf-8");

});

Done!

Skenario 4: Update Note (Memperbarui data catatan)

~~Pastikan response memiliki status code 200.~~

~~Pastikan header response Content-Type memiliki nilai application/json.~~

Pastikan body response adalah object.

Pastikan body response memiliki properti dan nilai yang sesuai.

Ketika mengakses catatan yang diperbaharui

Pastikan catatan yang diperbarui memiliki nilai terbaru.

Selanjutnya kita uji body response bahwa ia harus merupakan sebuah objek. Silakan buat test baru dengan nama “‘response body should be an object”.

pm.test('response body should be an object', () => {

});

Di dalamnya, dapatkan nilai response body sebagai JSON objek.

pm.test('response body should be an object', () => {

const responseJson = pm.response.json();

});

Kemudian, uji responseJson bahwa ia harus sebuah objek.

pm.test('response body should be an object', () => {

const responseJson = pm.response.json();

pm.expect(responseJson).to.be.an('object');

});

Done!

Skenario 4: Update Note (Memperbarui data catatan)

~~Pastikan response memiliki status code 200.~~

~~Pastikan header response Content-Type memiliki nilai application/json.~~

~~Pastikan body response adalah object.~~

Pastikan body response memiliki properti dan nilai yang sesuai.

Ketika mengakses catatan yang diperbaharui

Pastikan catatan yang diperbarui memiliki nilai terbaru.

Lanjut, uji nilai objek body response-nya. Buat test baru dengan nama “‘response body should have correct properti and value”.

pm.test('response body should have correct property and value', () => {

});

Di dalamnya, dapatkan nilai body response dalam bentuk JSON objek.

pm.test('response body should have correct property and value', () => {

const responseJson = pm.response.json();

});

Kemudian uji responseJson, ia harus memiliki properti dan nilai yang sesuai.

pm.test('response body should have correct property and value', () => {

const responseJson = pm.response.json();

pm.expect(responseJson).to.have.ownProperty('status');

pm.expect(responseJson.status).to.equals('success');

pm.expect(responseJson).to.have.ownProperty('message');

pm.expect(responseJson.message).to.equals('Catatan berhasil diperbarui');

});

Done!

Skenario 4: Update Note (Memperbarui data catatan)

~~Pastikan response memiliki status code 200.~~

~~Pastikan header response Content-Type memiliki nilai application/json.~~

~~Pastikan body response adalah object.~~

~~Pastikan body response memiliki properti dan nilai yang sesuai.~~

Ketika mengakses catatan yang diperbaharui

* Pastikan catatan yang diperbarui memiliki nilai terbaru.

Terakhir, bagaimana cara memastikan catatan yang diperbaharui memiliki nilai terbaru? Tentu caranya adalah dengan melakukan permintaan untuk mendapatkan catatan secara spesifik.

Karena kita sudah menguji dan memastikan bahwa permintaan tersebut berjalan dengan baik melalui skenario ke-3, sekarang kita bisa percaya diri dalam menggunakannya untuk memastikan data catatan pada skenario ini benar-benar telah diperbaharui.

Yuk buat test baru dengan nama “when request the updated note”.

pm.test('when request the updated note', () => {

});

Di dalamnya, kita lakukan permintaan untuk mendapatkan catatan secara spesifik berdasarkan id.

Dapatkan dulu nilai noteId dari environment variable.

pm.test('when request the updated note', () => {

const noteId = pm.environment.get('noteId');

});

Lalu buatlah request ke http://localhost:5000/notes/${notesId} dengan menggunakan method pm.sendRequest().

pm.test('when request the updated note', () => {

const noteId = pm.environment.get('noteId');

pm.sendRequest(`http://localhost:5000/notes/${noteId}`, (error, response) => {

});

});

Anda bisa lihat kode request di atas, method sendRequest() menerima dua parameter, yakni request URL dan fungsi response callback. Response callback akan terpanggil ketika permintaan sudah ditanggapi oleh response ataupun mengalami eror.

Eror dapat terjadi karena berbagai hal, salah satunya tidak adanya internet. Ketika eror, parameter error yang berada di response callback akan terisi nilainya. Namun bila permintaan berhasil dilakukan dan mendapatkan response dari server, maka parameter response-lah yang terisi nilainya.

Pola ini disebut dengan error first, di mana callback function mendahulukan parameter error dibandingkan dengan ketika operasi berhasil dijalankan. Node.js banyak menerapkan pola ini pada sebuah fungsi callback.

Yuk kita lanjutkan. Di dalam fungsi callback, kita buat test baru dengan nama “then the updated note should contain the latest data” untuk menguji nilai dari body response. Namun ingat, lakukan hal tersebut bila nilai error tidak terisi saja.

pm.test('when request the updated note', () => {

const noteId = pm.environment.get('noteId');

pm.sendRequest(`http://localhost:5000/notes/${noteId}`, (error, response) => {

if(!error) {

pm.test('then the updated note should contain the latest data', () => {

});

}

});

});

Selanjutnya di dalam test tersebut, kita dapatkan nilai objek note pada data melalui response body dalam bentuk JSON.

pm.test('when request the updated note', () => {

const noteId = pm.environment.get('noteId');

pm.sendRequest(`http://localhost:5000/notes/${noteId}`, (error, response) => {

if(!error) {

pm.test('then the updated note should contain the latest data', () => {

const responseJson = response.json();

const { data: { note } } = responseJson;

});

}

});

});

Sebelum menguji properti dan nilai dari objek note, tentukan nilai ekspektasi dari title, tags, dan body. Bila Anda lupa, Anda bisa lihat pada body request Update Note.

pm.test('when request the updated note', () => {

const noteId = pm.environment.get('noteId');

pm.sendRequest(`http://localhost:5000/notes/${noteId}`, (error, response) => {

if(!error) {

pm.test('then the updated note should contain the latest data', () => {

const responseJson = response.json();

const { data: { note } } = responseJson;

const expectedTitle = 'Catatan A Revised';

const expectedTags = ['Android', 'Web'];

const expectedBody = 'Isi dari Catatan A Revised';

});

}

});

});

Nice! Sekarang kita uji nilai dari properti title, tags, dan body yang dimiliki objek note dengan nilai ekspektasi.

pm.test('when request the updated note', () => {

const noteId = pm.environment.get('noteId');

pm.sendRequest(`http://localhost:5000/notes/${noteId}`, (error, response) => {

if(!error) {

pm.test('then the updated note should contain the latest data', () => {

const responseJson = response.json();

const { data: { note } } = responseJson;

const expectedTitle = 'Catatan A Revised';

const expectedTags = ['Android', 'Web'];

const expectedBody = 'Isi dari Catatan A Revised';

pm.expect(note.title).to.equals(expectedTitle);

pm.expect(note.tags).to.eql(expectedTags);

pm.expect(note.body).to.equals(expectedBody);

});

}

});

});

Done!

Skenario 4: Update Note (Memperbarui data catatan)

~~Pastikan response memiliki status code 200.~~

~~Pastikan header response Content-Type memiliki nilai application/json.~~

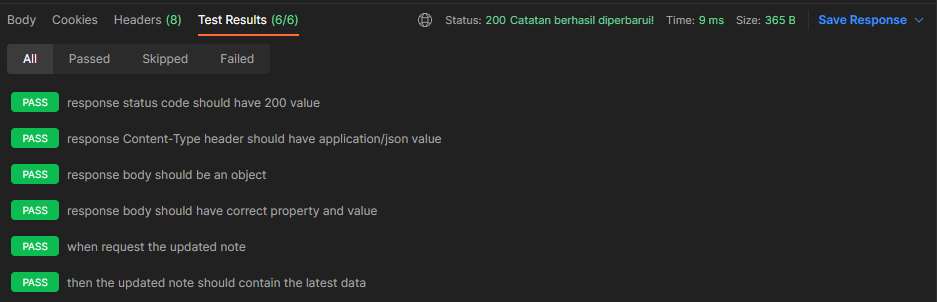
~~Pastikan body response adalah object.~~

~~Pastikan body response memiliki properti dan nilai yang sesuai.~~

~~Ketika mengakses catatan yang diperbaharui~~

* ~~Pastikan catatan yang diperbarui memiliki nilai terbaru.~~

Skenario 4 selesai! Yuk kita coba kirim permintaan dengan klik tombol Send.



Boom! Seluruh pengujian berhasil dijalankan.

Simpan request dengan menekan tombol Save dan ayo kita lanjutkan ke skenario terakhir.