**Lab: testen van een functie met dependencies in Jest**

In dit lab wordt uitgelegd hoe je een functie kunt unit testen met Jest. De functies die door de te testen functie worden aangeroepen, worden allen ge-mocked*\**.

Zoals je zult zien is er een ‘trucje’ nodig om de functies te mocken die binnen dezelfde module zijn gedefinieerd als waarin de te testen functie staat.

*\*Een mock functie vervangt als het ware de echte implementatie door een dummy implementatie. Dit heeft als voordeel dat de tijdens het testen de uitkomst van deze functie altijd hetzelfde zal zijn, omdat deze vooraf is gespecificeerd (dit in tegenstelling tot de uitkomst van de echte functie, die kan/zal variëren tijdens runtime)*

Voordat je deze lab kunt doorlopen moet je het volgende hebben geïnstalleerd:

1. Node.js, ga naar <https://nodejs.org/en/> als je dit nog niet hebt gedaan.
2. Een IDE
   1. bijvoorbeeld Visual Studio code, met tevens de extensies ‘vscode-jest’ en ‘vscode-jest-runner’
3. Het ‘NotifierApplication’ project, waarvoor je in dit lab testen gaat schrijven. Het is hier te downloaden: <https://github.com/AMIS-Services/sig-javascript-unittesting>

**Stap 0.**

Open het ‘NotifierApllication’ project in jouw IDE.

**Stap 1.**

Voeg de volgende development dependencies toe aan de package.json file (na het blok dependencies):

  "devDependencies": {

    "nodemon": "^2.0.12",

    "babel-core": "^7.0.0-beta.3",

    "babel-jest": "^27.0.0-next.8",

    "babel-preset-env": "^7.0.0-beta.3",

    "esm": "^3.2.25",

    "jest": "^27.0.0-next.8"

  }

**Stap 2.**

In de ‘processor’ folder staat de file subscriptionProcessor.js.

Deze file bevat een functie met de naam getProcessorsForActiveSubscriptions en heeft de volgende definitie:

function getProcessorsForActiveSubscriptions() {

    activeList = subscriptionService.getActive("Api01");

    channels = determineChannelMap(activeList);

    return = createChannelProcessors(channels, activeList);

}

Deze functie getProcessorsForActiveSubscriptions roept als eerste een functie getActive aan die is opgenomen in de module /NotificationApplication/services/subscriptions.js.

Hierna worden de functies determineChannelMap en createChannelProcessors aangeroepen. Deze staan beiden in dezelfde module als waarin getProcessorsForActiveSubscriptions is opgenomen.

Voor deze functie getProcessorsForActiveSubscriptions gaan we in de volgende stappen een test met Jest maken.

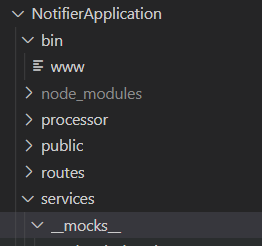
**Stap 3.**

Bekijk de inhoud van de module subscriptions.js in de services folder. Zie dat de functie getActive een lijst met de naam ‘subscriptionList’ filtert en het resultaat daarvan afgeeft.

Voorafgaand aan de filtering bevat de lijst 2x de naam Jan en 2x de naam Piet.

Voor de module ‘subscriptions.js’ gaan we nu een mock schrijven.

Maak een nieuwe sub folder onder de services folder met de naam ‘\_\_mocks\_\_’.



Maak in deze ‘\_\_mocks\_\_’ subfolder een nieuw file met de naam subscriptions.js aan.

Kopieer de volgende inhoud en plak het in deze file.

const subscriptions = jest.createMockFromModule('./subscriptions');

const subscriptionList = [{

    id: '1',

    api: '1',

    subscriber: 'John',

    clientId: '1',

    sessionId: '111111111',

    channelId: '1',

    subscribedAt: '2021-06-19T18:00:00',

    unsubscribedAt: null,

    sessionRemovedAt: null

},

{

    id: '2',

    api: '1',

    subscriber: 'Jane',

    clientId: '1',

    sessionId: '999999999',

    channelId: '1',

    subscribedAt: '2021-06-18T21:02:00'

}];

function getActive (channel) {

    return subscriptionList;

}

module.exports = {

    getActive

}

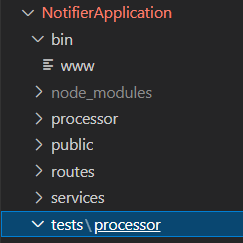
De ‘\_\_mocks\_\_’ folder bevat hiermee nu een mock voor de file subscriptions.js in de services folder.

Merk op dat deze ‘gemockte’ functie getActive een subscriptionList met 2 elementen als resultaat geeft en dat deze de namen John en Jane bevat (in plaats van de namen Jan en Piet in de echte implementatie van deze functie).

**Stap 4.**

Maak in de folder ‘NotifierApplication’ een nieuwe sub folder met de naam ‘tests’ aan.

Maak vervolgens daarin de subfolder ‘processor’ aan.



Maak nu in de zojuist gemaakte subfolder ‘processor’ een nieuwe file aan met de naam ‘subscriptionProcessor.test.js’ aan en geef het de volgende inhoud.

subscriptionProcessor = require('../../processor/subscriptionProcessor');

jest.mock('../../services/subscriptions');

test('test', () => {

    subscriptionProcessor.determineChannelMap = jest.fn().mockReturnValue(new Map());

    subscriptionProcessor.createChannelProcessors = jest.fn().mockReturnValue(new Map());

    let result = subscriptionProcessor.getProcessorsForActiveSubscriptions();

    console.log(result);

    //verify the outcome

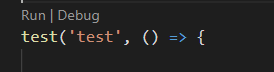
    expect(result).toEqual(new Map());

});

De regel met de instructie ‘jest.mock(../../services/subscriptions)’ zorgt er voor dat de mock die eerder hierboven is geplaatst in de ‘\_\_mocks\_\_’ folder gebruikt wordt als een vervanger voor de echte subscriptions module.

**Stap 5.**

Probeer nu deze test te runnen door in Visual Studio Code de ‘Run|Debug’ optie te kiezen. Of door in een shell op de command line de instructie ‘npm test’ te geven.



Stap 6.

Zoals je ziet faalt deze test. Dit komt doordat de functies determineChannelMap, createChannelProcessors en getProcessorsForActiveSubscriptions de private scope hebben in de file NotifierApplication/processor/subscriptionProcessor. Dit kan worden opgelost door het ‘module.exports’ blok in deze file als volgt uit te breiden:

module.exports = {

    sendNotifications,

    getProcessorsForActiveSubscriptions,

    determineChannelMap,

    createChannelProcessors

}

Stap 7.

Run de test opnieuw.

Merk op in de console dat de eerder gemaakte mock voor subscriptions.js inderdaad is gebruikt (te zien aan de getoonde namen John en Jane).

Maar de mocks die zijn gespecificeerd voor de methoden determineChannelMap en createChannelProcessor zijn echter niet gebruikt, want als dat wel het geval zou zijn dan was het resultaat een lege map geweest (dus ipv een resultaat met John en Jane).

De reden hiervoor is dat this in de module met de functies niet verwijst naar de mocks.

Stap 8.

Om er voor te zorgen dat de mocks voor de methoden determineChannelMap en createChannelProcessor worden gebruikt moeten 3 kleine aanpassingen worden gedaan.

1. Voeg in de file NotifierApplication/processor/subscriptionProcessor de volgende constante toe direct boven het module.exports blok:

const exportFunctions = {

    getProcessorsForActiveSubscriptions,

    determineChannelMap,

    createChannelProcessors

};

1. Wijzig tevens de module.exports als volgt:

module.exports = {

    sendNotifications,

    exportFunctions

    // getProcessorsForActiveSubscriptions,

    // determineChannelMap,

    // createChannelProcessors

}

De functies die eerder publiek gemaakt zijn, zijn dat nu indirect via exportFunctions. Pas de getProcessorsForActiveSubscriptions daarom als volgt aan:

function getProcessorsForActiveSubscriptions() {

    activeList = subscriptionService.getActive("Api01");

    channels = exportFunctions.determineChannelMap(activeList);

    return exportFunctions.createChannelProcessors(channels, activeList);

}

1. In de file subscriptionProcessor.test.js moet in de test eveneens deze indirectie worden opgenomen door als volgt de ‘exportFunctions’ op te nemen.

test('test', () => {

    subscriptionProcessor.exportFunctions.determineChannelMap = jest.fn().mockReturnValue(new Map());

    subscriptionProcessor.exportFunctions.createChannelProcessors = jest.fn().mockReturnValue(new Map());

    let result = subscriptionProcessor.exportFunctions.getProcessorsForActiveSubscriptions();

    console.log(result);

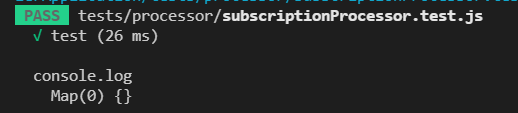
    //verify the outcome

    expect(result).toEqual(new Map());

});

Stap 8.

Run de test opnieuw en zie in de console dat nu wel de mock gebruikt is die een lege map als resultaat geeft.



**Deel 2 van dit lab.**

Je hebt nu als het goed is gezien hoe je de functies in een module kunt mocken die door een te testen functie in deze zelfde module gebruikt worden. Ook heb je kunnen zien hoe je een complete afzonderlijke module mockt, en die wordt geïmporteerd in de module met de te testen functie.

In dit tweede deel ga je zien hoe een klasse met functies mockt en hoe je van deze klasse een functie kunt unit testen.

**Stap 9.**

Maak een nieuwe file aan in de folder NotifierApplication/processor en noem deze SubscriptionProcessorAsClass.js.

Geef het de onderstaande inhoud:

const subscriptionService = require("../services/subscriptions");

const vaultService = require("../services/vault");

const ChannelProcessor = require("./channelProcessor");

const Channel = require("./channel");

module.exports = class SubscriptionProcessor {

    channelProcessors;

    channels;

    activeList;

    constructor() {

        this.channelProcessors = new Map();

        this.channels = new Map();

    }

    determineChannelMap(subscriptions) {

        let channels = new Map();

        subscriptions.forEach(subscription => {

            secret = subscription['channelId'];

            subscriber = subscription['subscriber'];

            key = subscriber + "|" + secret;

            if (!channels.get(key)) {

                channel = new Channel(secret, subscriber, vaultService.getSecret(secret));

                channels.set(key, channel);

            }

        });

        return channels;

    }

    getProcessorsForActiveSubscriptions() {

        var activeList = subscriptionService.getActive("Api01");

        var channels = this.determineChannelMap(activeList);

        return this.createChannelProcessors(channels, activeList);

    }

    getProcessorsForAddedSubscriptions() {

        mostRecent = determineMostRecent(channelProcessors.keys());

        addedList = subscriptionService.getAdded("Api01", mostRecent);

        channels = determineChannelMap(addedList);

        return createChannelProcessors(channels, addedList);

    }

    determineMostRecent(subscriptions) {

        mostRecentSub = subscriptions.next();

        maxValue = new Date(mostRecentSub['subscribedAt']);

        for (const subscription of subscriptions) {

            currValue = new Date(subscription['subscribedAt']);

            if (currValue > maxValue) {

                maxValue = currValue;

                mostRecentSub = subscription;

            }

        }

        return mostRecentSub;

    }

    createChannelProcessors(channels, subscriptions) {

        result = new Map();

        subscriptions.forEach(subscription => {

            channel = channels.get(subscription.subscriber + "|" + subscription.channelId);

            if (channel != null) {

                proc = new ChannelProcessor(channel);

                result.set(subscription, proc);

            }

        });

        return result;

    }

    sendNotifications() {

        channelProcessors = getProcessorsForActiveSubscriptions();

        added = getProcessorsForAddedSubscriptions();

        if (added.size > 0) {

            added.forEach((value, key) => {

                channelProcessors.set(key, value);

            });

        }

        if (channelProcessors.size > 0) {

            channelProcessors.forEach((channelProcessor) => {

                console.log("channel = " + channelProcessor.channel.channelId);

                channelProcessor.run();

            });

        }

        return;

    }

}

Zoals je waarschijnlijk al hebt gezien is dit een klasse waarin dezelfde functies zijn opgenomen als in de module subscriptionProcessor.js (met aan paar kleine tweaks om het geheel valide te maken).

**Stap 10.**

Maak in de folder NotifierApplication/tests/processor een nieuwe file subscriptionProcessorAsClass.test.js

Voeg als eerste een import voor de klasse SubscriptionProcessorAsClass toe aan deze file

SubscriptionProcessorAsClass = require("../../processor/SubscriptionProcessorAsClass");

En voeg daarna de volgende test toe:

test('test', () => {

    jest.mock('../../services/subscriptions');

    //

    const processor = new SubscriptionProcessorAsClass();

    jest.spyOn(SubscriptionProcessorAsClass.prototype, 'determineChannelMap').mockImplementation(function() {return new Map();});

    jest.spyOn(SubscriptionProcessorAsClass.prototype, 'createChannelProcessors').mockImplementation(function() {return new Map();});

    //invoke the test

    let result = processor.getProcessorsForActiveSubscriptions();

    console.log(result);

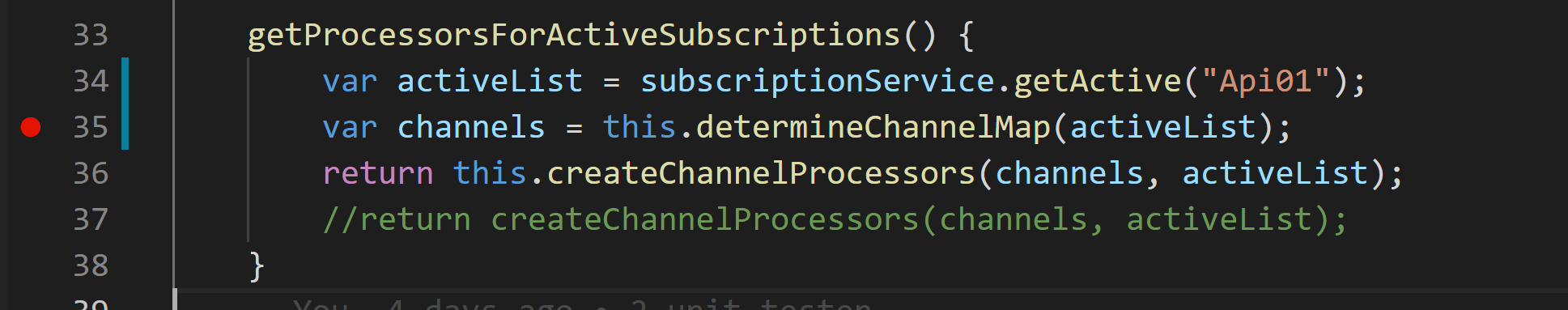
    //verify the outcome

    expect(result).toEqual(new Map());

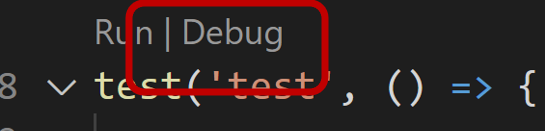
} );

Run deze test en zie dat deze valide lijkt te zijn.

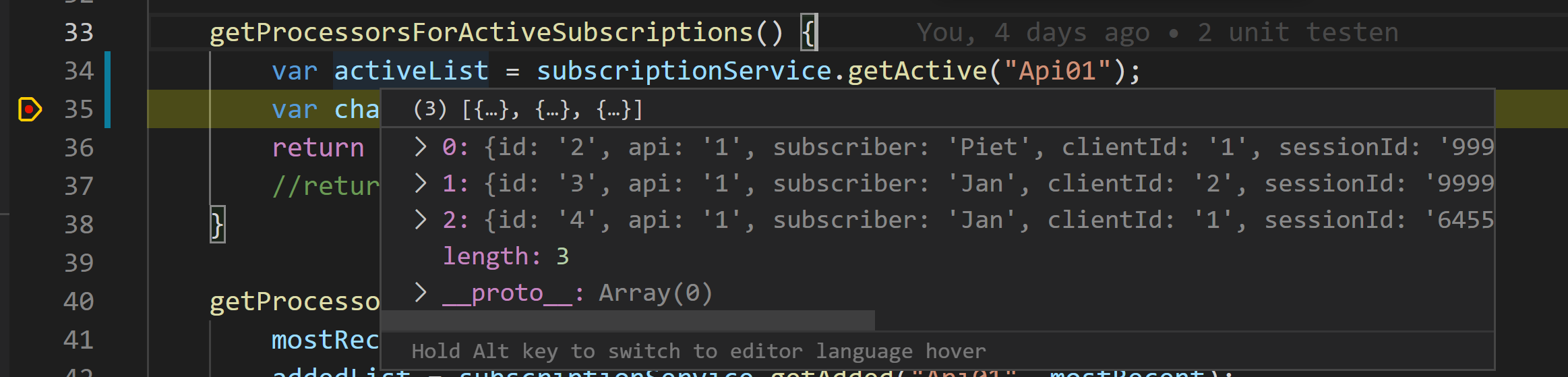
Laten we echter eens kijken welke response de aanroep van subscriptionService.getActive(“Api01”) geeft. Plaats hiervoor een breakpoint in de functie zodat de waarde van activeList kan worden bekeken nadat deze is toegekend.



Debug de test in subscriptionProcessorAsClass.test.js:

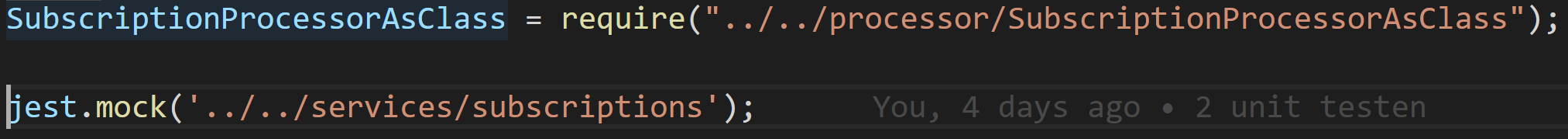


Zie dat activeList niet de gespecificeerde waarden uit de mock heeft. De mock heeft immers John en Jane als namen. Blijkbaar is de echte implementatie van subscriptionService uitgevoerd.



**Stap 11.**

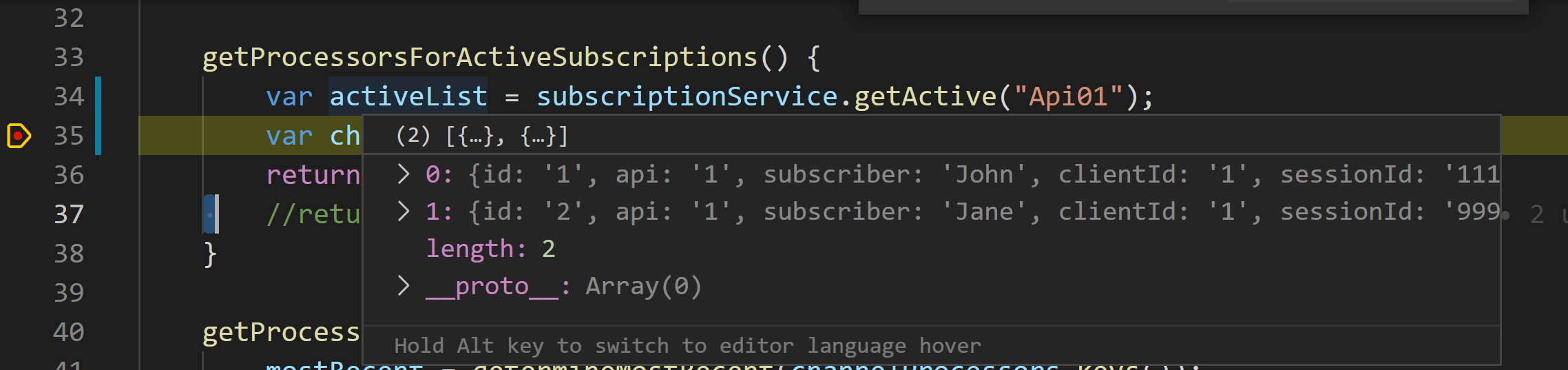
Om dit te fixen moet in de file subscriptionProcessorAsClass.test.js het statement: ‘jest.mock('../../services/subscriptions');’ worden verplaatst naar de globale scope, en wel direct na het import statement van de klasse SubscriptionProcessorAsClass. Dus als volgt:



De import die in de test methode staat moet worden/zijn verwijderd:



Debug nu opnieuw de test en zie of nu wel de mock wordt toegepast. Als het goed is zie je nu wel de namen John en Jane als volgt:



Zou je deze controle in de test kunnen vastleggen?

(op het moment van schrijven heb ik nog niet gevonden hoe dat moet B2W)

**Testen van de sendNotifications methode**

In deze test gaat de volledige sendnotifications getest worden, waarbij de modules subscriptions.js en notifications.js gemocked gaan worden. Als je naar de implementatie van sendNotifications kijkt dan zie je dat deze twee methodes aanroept getProcessorsForActiveSubscriptions en getProcessorsForAddedSubscriptions. Deze twee methodes roepen een methode in subscriptions.js aan. Deze gaan we mocken. Daarnaast wordt in de methode sendNotifications channelProcessor.run(); aangeroepen. Als we in de module ChannelProcessor.js kijken zien we dat de run methode de getNotifications en sendNotification op de notification module aanroept. Ook deze twee methodes gaan we mocken.

**Stap 1**

Maak een nieuwe test bestand aan in tests/processor met de naam subscriptionProcessor.sendNotifications.test.js

In de test subscriptionProcessor.test.js zijn drie requires nodig:

const subscriptionService = require('../../services/subscriptions');

const notificationService = require('../../services/notifications');

const processor = require('../../processor/subscriptionProcessor');

In deze test voeg je twee mocks toe:

jest.mock('../../services/subscriptions');

jest.mock('../../services/notifications');

Hierbij valt op dat we nu geen mock implementatie van notifications.js in de \_\_mocks\_\_ folder plaatsen. Hierdoor genereert Jest een automatische mock, waarbij alle methodes vervangen worden met mock functions die een undefined retourneren.

**Stap 2**

Maak een nieuwe lege test aan die je een mooie naam geeft om de sendnotifications methode te testen, bijv:

test('send notifications', () => {

});

**Stap 3**

Deze test gaat geen gebruik maken van de mock die gedefinieerd staat in de \_\_mocks\_\_/subscriptions.js. In de zojuist aangemaakte test zet je de volgende twee lijsten:

    const testlist = [{

        id: '1',

        api: '1',

        subscriber: 'Hein',

        clientId: '1',

        sessionId: '111111111',

        channelId: '1',

        subscribedAt: '2021-06-19T18:00:00',

        unsubscribedAt: '2021-06-20T19:00:00',

        sessionRemovedAt: '2021-06-20T20:00:00'

    },

    {

        id: '2',

        api: '1',

        subscriber: 'Suzanne',

        clientId: '55',

        sessionId: '87776',

        channelId: '2',

        subscribedAt: '2021-06-19T18:00:00',

        unsubscribedAt: '2021-06-20T19:00:00',

        sessionRemovedAt: '2021-06-20T20:00:00'

    }];

    subscriptionService.getActive = jest.fn();

    subscriptionService.getAdded = jest.fn();

    const mockGetActive = subscriptionService.getActive.mockReturnValue(testlist);

    const notificationList1 = [{

        id: '1',

        channelId: '1',

        message: 'message1',

        createdAt: '2021-06-18T21:02:00'

    },

    {

        id: '2',

        channelId: '1',

        message: 'message2',

        createdAt: '2021-07-18T21:02:00'

    }]

    const notificationList2 = [

    {

        id: '2',

        channelId: '2',

        message: 'message3',

        createdAt: '2021-07-18T21:02:00'

    }]

Zoals je ziet zijn de namen in deze lijst Hein en Suzanne, wat weer andere namen zijn dan in de daadwerkelijke implementatie en dan in de \_\_mocks\_\_ folder staat.

**Stap 4**

Om de subscriptions module gemocked te krijgen voeg je de volgende code toe aan de test:

    subscriptionService.getActive = jest.fn();

    subscriptionService.getAdded = jest.fn();

    const mockGetActive = subscriptionService.getActive.mockReturnValue(testlist);

jest.fn() zorgt ervoor dat er een mock functie wordt gezet op de getActive en getAdded methodes op de subscriptions module. In het geval van getActive overschrijft dit de bestaande mock methode die gedefinieerd staat in \_\_mocks\_\_/subscriptions.js. Hierdoor kunnen we op de derde regel aangeven dat we de testlist willen retourneren als getActive aangeroepen wordt van deze mock. We stoppen deze mockmethode in een const zodat we straks expectations kunnen definieren op de aanroep naar getActive.

Om de notifications module te mocken voeg je de volgende code toe aan de test

const mockGetNotifications = notificationService.getNotifications.mockImplementation((x, y) => { if (x === '1') {return notificationList1;} else { return notificationList2;}});

const mockSendNotification = notificationService.sendNotification;

Door gebruik te maken van mockimplementation kan je aangeven welke notificationlist we willen retourneren als getNotifications aangeroepen wordt met een bepaalde parameter (x en y zijn de twee parameters van getNotifications). getNotifications wordt namelijk twee keer aangeroepen met twee verschillende parameters. Namelijk 1, welke de channelId is van subscriber Hein, en 2 welke de channelId is van Suzanne.

**Stap 5**

Nu de benodigde mocks klaar staan kan je de methode aanroepen in de test:

subscriptionProcessor.sendNotifications();

Doe een ‘npm test’. De test zal succesvol zijn.

**Stap 6**

Nu kan je expectations zetten op de mocks die zijn aangeroepen. Je kan bijvoorbeeld controleren of mocks aangeroepen zijn door het volgende toe te voegen aan de test:

    expect(mockGetActive).toHaveBeenCalled();

    expect(mockGetNotifications).toHaveBeenCalled();

    expect(mockGetActive).toHaveBeenCalled();

    expect(mockSendNotification).toHaveBeenCalled();

Daarnaast kan je ook controleren met welke waardes de gemockte methodes zijn aangeroepen:

    // Mock is called 3 times

    expect(mockSendNotification.mock.calls.length).toBe(3);

    // First argument of first call is Hein

    expect(mockSendNotification.mock.calls[0][0]).toBe('Hein');

    // Second argument of first call is first element of notificationList1

    expect(mockSendNotification.mock.calls[0][1]).toEqual(notificationList1[0]);

    // First argument of second call is Hein

    expect(mockSendNotification.mock.calls[1][0]).toBe('Hein');

    // Second argument of second call is second element of notificationList1

    expect(mockSendNotification.mock.calls[1][1]).toEqual(notificationList1[1]);

    // First argument of third call is Suzanne

    expect(mockSendNotification.mock.calls[2][0]).toBe('Suzanne');

    // Second argument of third call is first element of notificationList2

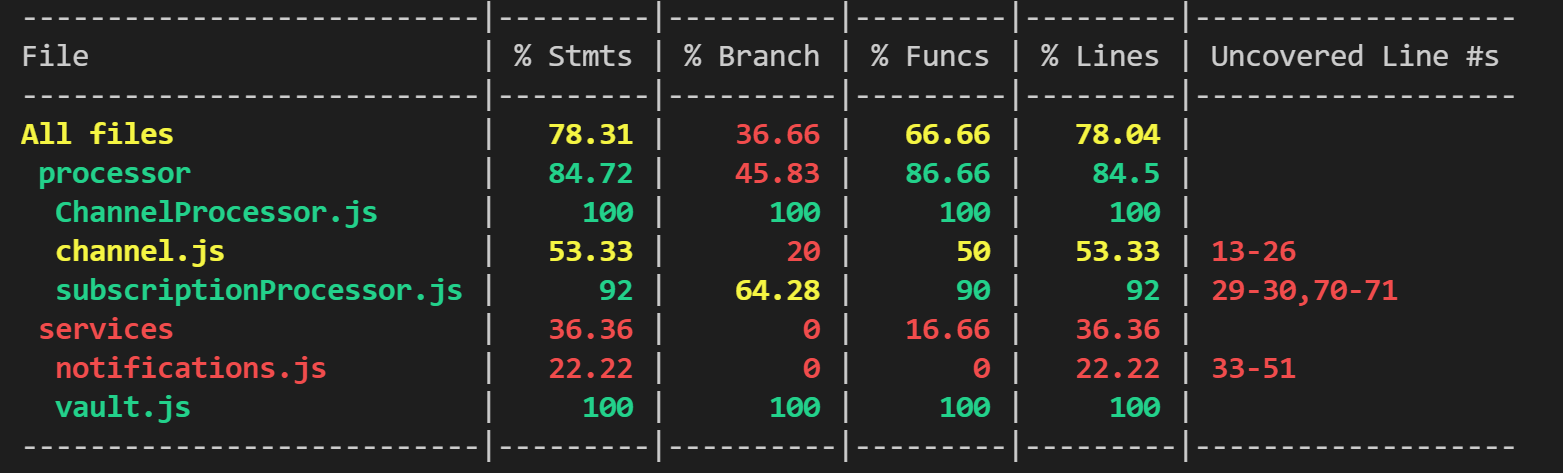
    expect(mockSendNotification.mock.calls[2][1]).toEqual(notificationList2[0]);

**Coverage**

Om de code coverage te tonen van je tests zet je --coverage achter jest in de package.json:

"test": "jest --coverage"

Als je nu een ‘npm test’ runt dan krijg je een overzicht waarin je de percentages coverage per klasse ziet en welke regels nog niet gecovered zijn door tests:



**Volledige coverage bereiken**

Met de volgende testen gaan we alle scenarios af die we in de vorige testen niet hebben gehad, om zo 100% coverage te hebben.

**Stap 1**

Laten we beginnen met de subscriptionProcessor.js. Deze zit op 92% statement en line coverage. Regels 29, 30, 70 en 71 worden niet geraakt in onze huidige testen.

Als we naar regels 70 en 71 kijken, zien we het volgende stukje code:

if (added.size > 0) {

        added.forEach((value, key) => {

            channelProcessors.set(key, value);

        });

Dit stukje bereiken we alleen als added groter is dan 0.

**Stap 2**

added wordt bepaald door getProcessorsForAddedSubscriptions(). Eerder hebben we geen return waarde gegeven aan de gemockte getAdded, waardoor er hier 0 uit komt.

Door deze nu wel een return waarde te geven, kunnen we regels 70 en 71 bereiken.

    subscriptionService.getAdded = jest.fn();

    const mockGetActive = subscriptionService.getActive.mockReturnValue(testlist);

    const mockGetAdded = subscriptionService.getAdded.mockReturnValue(testlist2);

**Stap 3**

We voegen ook een testlist2 toe naast de bestaande testlist, die wordt teruggegeven in de gemockte getAdded. Dit zal een subscriber zijn die sinds de laatste run is toegevoegd:

    const testlist = [{

…

    }];

    const testlist2 = [{

        id: '3',

        api: '1',

        subscriber: 'Kees',

        clientId: '34',

        sessionId: '233455',

        channelId: '4',

        subscribedAt: '2021-06-19T18:00:00',

        unsubscribedAt: '2021-06-20T19:00:00',

        sessionRemovedAt: '2021-06-20T20:00:00'

    }];

**Stap 4**

Voor deze nieuwe subscriber willen we ook dat er een notificatie wordt opgehaald. We voegen nog een notificationlist toe die matcht met het channelId, en breiden de mockGetNotifications uit:

    const notificationList2 = [

…

    }]

    const notificationList3 = [

        {

            id: '4',

            channelId: '4',

            message: 'message4',

            createdAt: '2021-07-18T21:02:00'

        }]

    const mockGetNotifications = notificationService.getNotifications.mockImplementation((x, y) => { if (x === '1') {return notificationList1;} else if (x === '2') {return notificationList2;} else { return notificationList3;}});

**Stap 5**

Omdat het nu ook belangrijk is om te weten of getAdded wel is aangeroepen, moet deze ook worden toegevoegd aan de expectations:

    expect(mockGetActive).toHaveBeenCalled();

    expect(mockGetAdded).toHaveBeenCalled();

    expect(mockGetNotifications).toHaveBeenCalled();

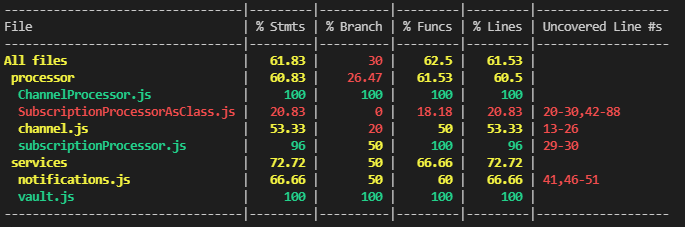
**Stap 6**

Dat betekent ook dat er een extra notificatie wordt verzonden, dus moet de expectation opgehoogd worden van 3 naar 4:

    // Mock is called 4 times

    expect(mockSendNotification.mock.calls.length).toBe(4);

Doe een ‘npm test’. De test zal succesvol zijn.



**Stap 7**

Laten we nu naar regels 29 en 30 kijken:

function determineMostRecent(subscriptions) {

    let mostRecentSub = subscriptions.next();

    let maxValue = new Date(mostRecentSub['subscribedAt']);

    for (const subscription of subscriptions) {

        currValue = new Date(subscription['subscribedAt']);

        if (currValue > maxValue) {

            maxValue = currValue;

            mostRecentSub = subscription;

        }

    }

    return mostRecentSub;

}

Hier itereer je over de subscriptions om daarmee de meest recente te bepalen.

**Stap 8**

Simpel, we hoeven alleen de testdata aan te passen, dat de eerste in de array een eerdere subscribedAt datum heeft, zodat wanneer we bij de tweede aankomen we in deze if statement landen. We passen het aan van 19 juni naar 18 juni:

    const testlist = [{

        id: '1',

        api: '1',

        subscriber: 'Hein',

        clientId: '1',

        sessionId: '111111111',

        channelId: '1',

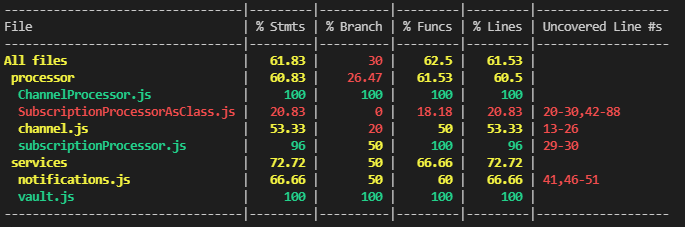
        subscribedAt: '2021-06-18T18:00:00',

        unsubscribedAt: '2021-06-20T19:00:00',

        sessionRemovedAt: '2021-06-20T20:00:00'

    },

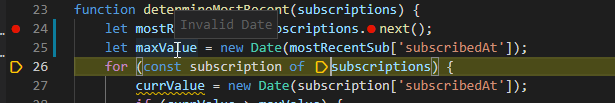
Doe een ‘npm test’. De test zal succesvol zijn.



**Stap 9**

De test is succesvol, maar de coverage is niet verhoogd. Regels 29 en 30 worden nog steeds niet bereikt. Wat is hier aan de hand? Zet een breakpoint op regel 24 in subscriptionProcessor.js en run de test in debug.

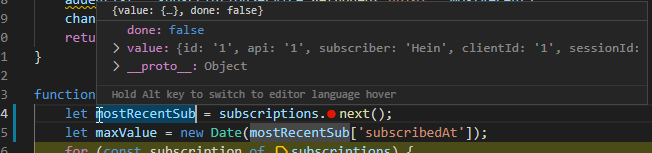
Als je de code debugt, zal je het volgende vinden:



maxValue is een Invalid Date. Hierdoor kan de if-statement niet uitgevoerd worden, omdat er niet mee vergeleken kan worden.

**Stap 10**

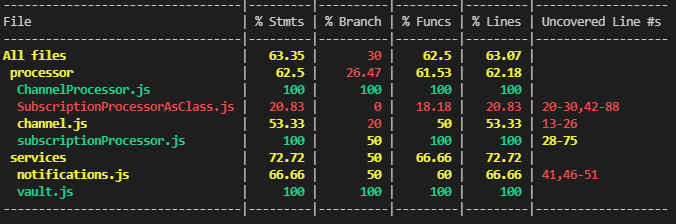
Als je verder debugged, zul je zien dat mostRecentSub incorrect is toegewezen:



Het is een iterator object, maar het zou de value van dat object moeten bevatten. Dat is makkelijk opgelost door regel 24 aan te passen:

    let mostRecentSub = subscriptions.next().value;

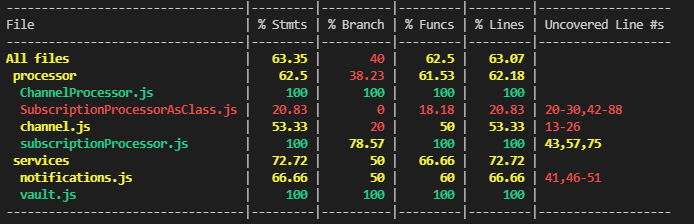
Doe een ‘npm test’. De test zal succesvol zijn.



Je zult zien dat de statement en line coverage op 100% zit.

De branch coverage is nog niet op 100%, hiermee wordt bedoeld of alle mogelijke branches zijn getest. Hiervoor moet je meerdere testen maken die de verschillende opties van if-statements nagaan.

Als je van alle aanpassingen in dit lab een aparte test zou maken en naast de test van het vorige lab zou draaien, zou je een hogere branch% zien:



**Clear mocks**

Als je meerdere testen maakt en draait, is het belangrijk dat je tussen alle testen door de mocks opschoont. Dat doe je door de volgende code toe te voegen vóór al je testcode:

afterEach(() => {

    jest.clearAllMocks();

  });

Als je dit niet doet, dan kan je tegen fouten aanlopen omdat sommige expectations worden beïnvloed door vorige test, zoals mock.calls.length, die zal dan door blijven tellen.