**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.

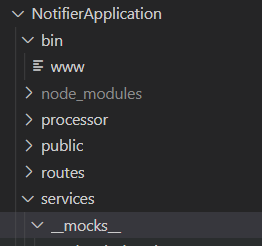
**Stap 3.**

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

}

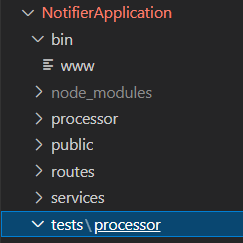
De ‘\_\_mocks\_\_’ folder bevat hiermee nu een mock voor de file subscriptions.js in de services folder.

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

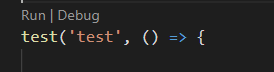
    expect(result).toEqual(new Map());

});

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);

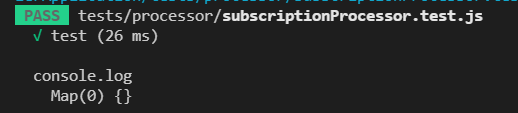
    //verify the outcome

    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);

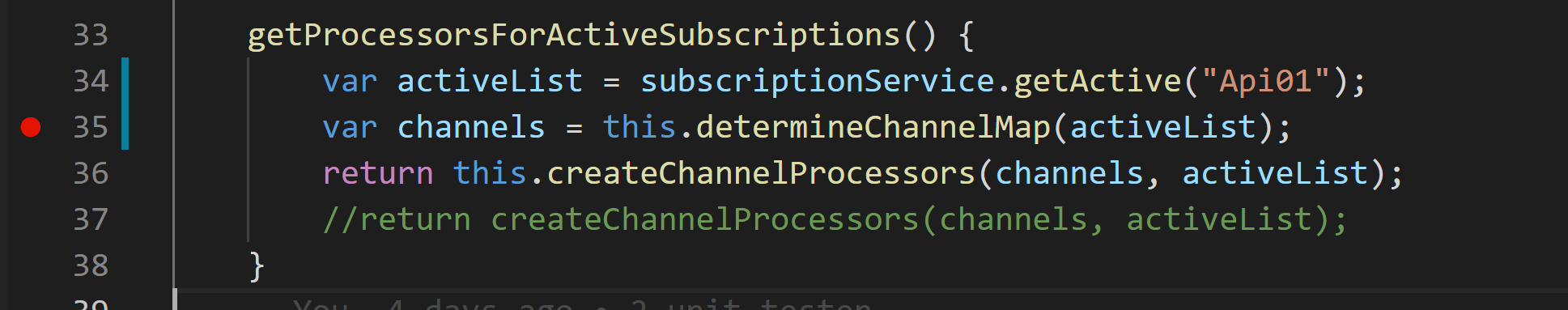
    //verify the outcome

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

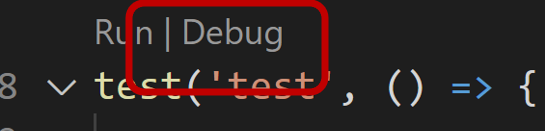
} );

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

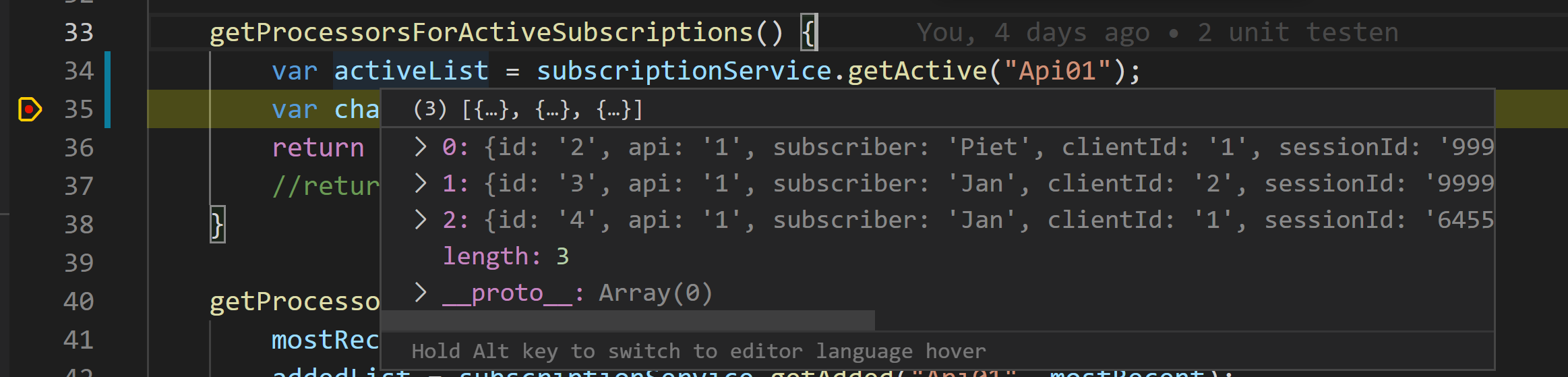
Laten we echter eens kijken welke response de aanroep van subscriptionService.getActive(“Api01”) geeft. Plaats hiervoor een breakpoint in de functie zodat de waarde van activeList kan worden bekeken nadat deze is toegekend.



Debug de test in subscriptionProcessorAsClass.test.js:

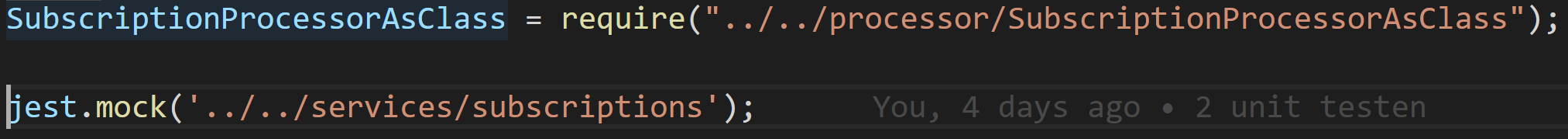


Zie dat activeList niet de gespecificeerde waarden uit de mock heeft. De mock heeft immers John en Jane als namen. Blijkbaar is de echte implementatie van subscriptionService uitgevoerd.



**Stap 11.**

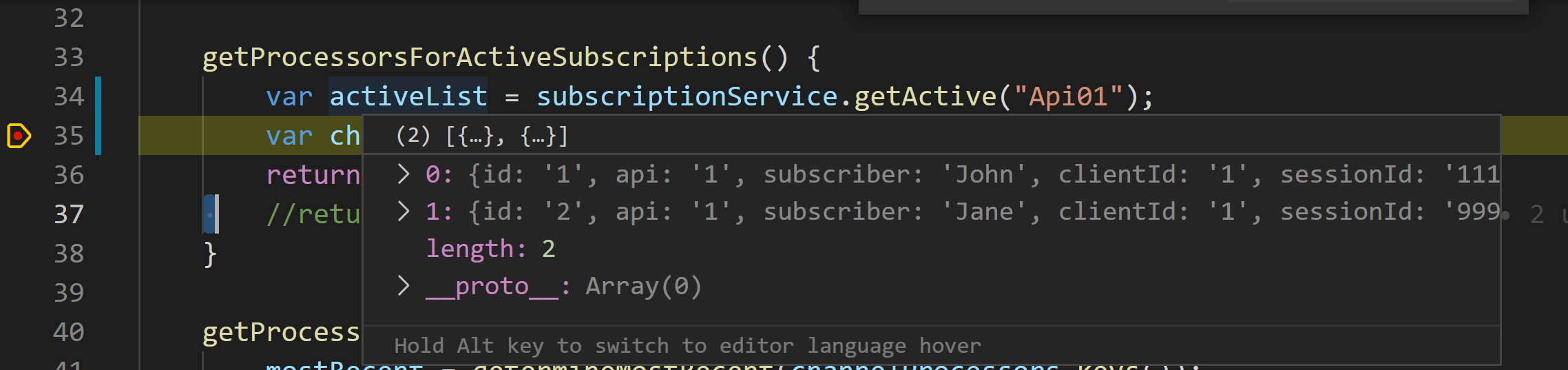
Om dit te fixen moet in de file subscriptionProcessorAsClass.test.js het statement: ‘jest.mock('../../services/subscriptions');’ worden verplaatst naar de globale scope, en wel direct na het import statement van de klasse SubscriptionProcessorAsClass. Dus als volgt:



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)