Workshop NodeJS for integration developers

Using NodeJS, Docker, MongoDB

SIG 28 augustus 2018

Inhoud

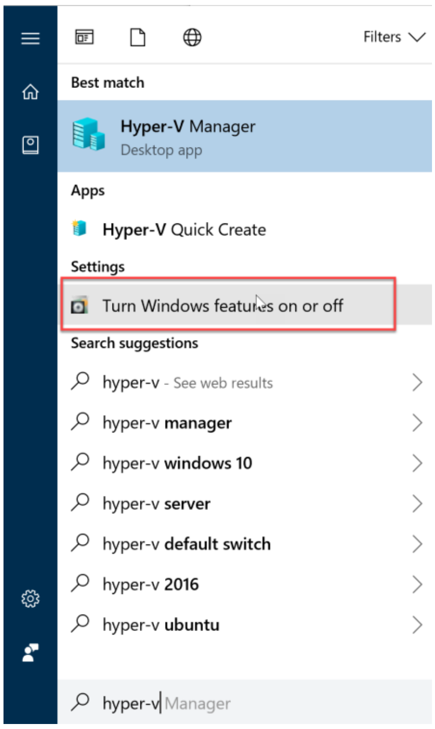
Prerequisites

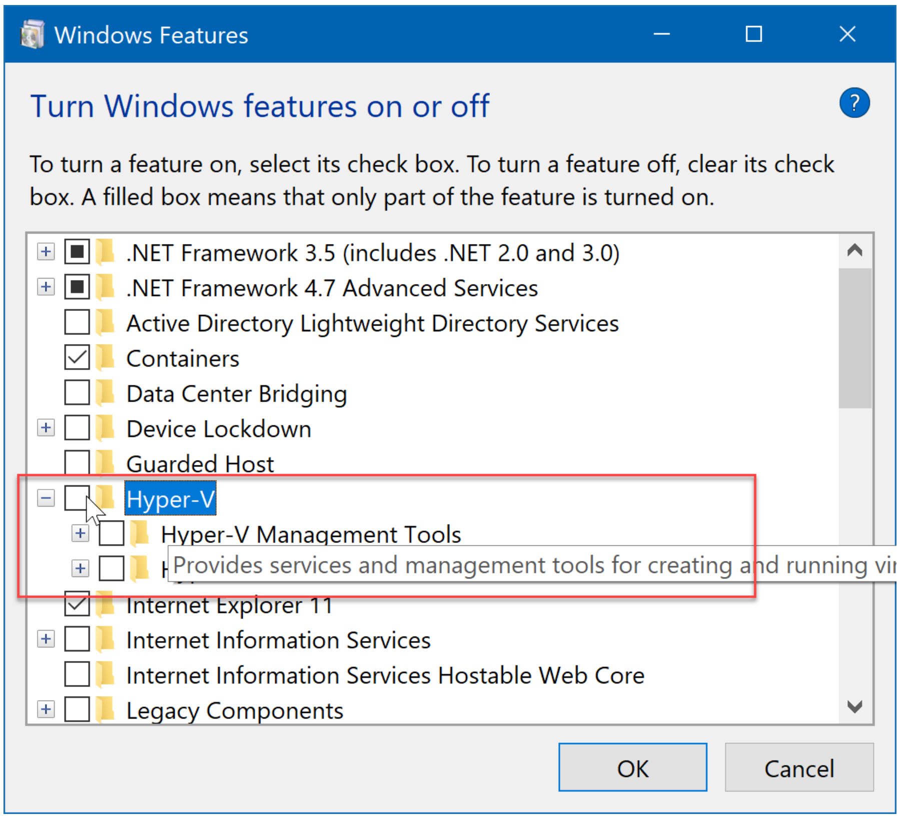
For following this workshop you need the following:

- VirtualBox 5.2.12 or later (VM has been tested with 5.2.16)
- Course VM

Make sure Virtualbox *5.2.12* or later is installed. Download and install it from www.virtualbox.org. If the version differs, the guest additions might not be able to communicate with the host OS. Issues you might encounter is that the bidirectional clipboard does not function.
The machine you're running the VM on requires at least 5Gb of available RAM and around 10Gb (preferably 30Gb) of free disk space.

Note that if you have Hyper-V enabled on your Windows 10 machine, you need to deactivate it before using Virtualbox.




Windows Features

Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- .NET Framework 3.5 (includes .NET 2.0 and 3.0)
- .NET Framework 4.7 Advanced Services
- Active Directory Lightweight Directory Services
- Containers
- Data Center Bridging
- Device Lockdown
- Guarded Host
- Hyper-V
  - Hyper-V Management Tools
  - Provides services and management tools for creating and running vi
- Internet Explorer 11
- Internet Information Services
- Internet Information Services Hostable Web Core
- Legacy Components

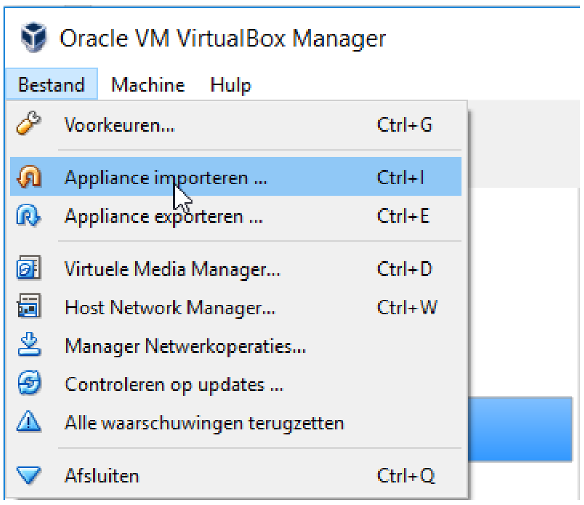
OK Cancel

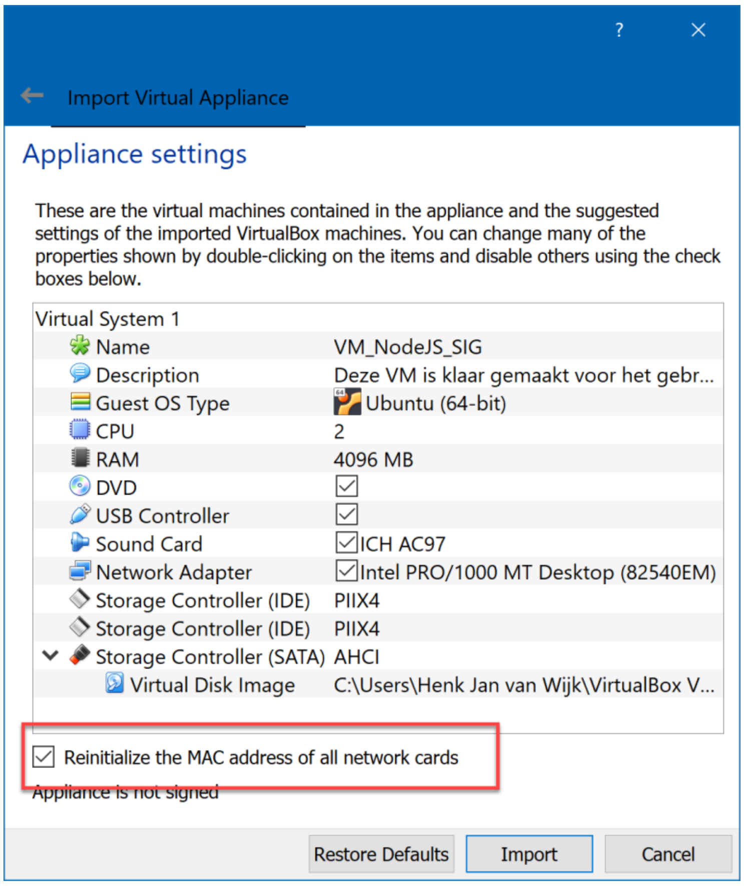
Your machine will reboot.

Importing the VM

Download the image file from: https://conclusionfutureit-my.sharepoint.com/:f:/g/personal/matthijs_breeman_amis_nl/EmkHnMTp0LVJigAoU8ZIZg0BQIljo_nDxTgaty_tPxyNNQ?e=kcJGqU *(2,7 GB)*

Import the OVA file in Virtualbox


Oracle VM VirtualBox Manager
Bestand Machine Hulp
Voorkeuren... Ctrl+G
Appliance importeren ... Ctrl+I
Appliance exporteren ... Ctrl+E
Virtuele Media Manager... Ctrl+D
Host Network Manager... Ctrl+W
Manager Netwerkoperaties...
Controleren op updates ...
Alle waarschuwingen terugzetten
Afsluiten Ctrl+Q



Appliance settings

These are the virtual machines contained in the appliance and the suggested settings of the imported VirtualBox machines. You can change many of the properties shown by double-clicking on the items and disable others using the check boxes below.

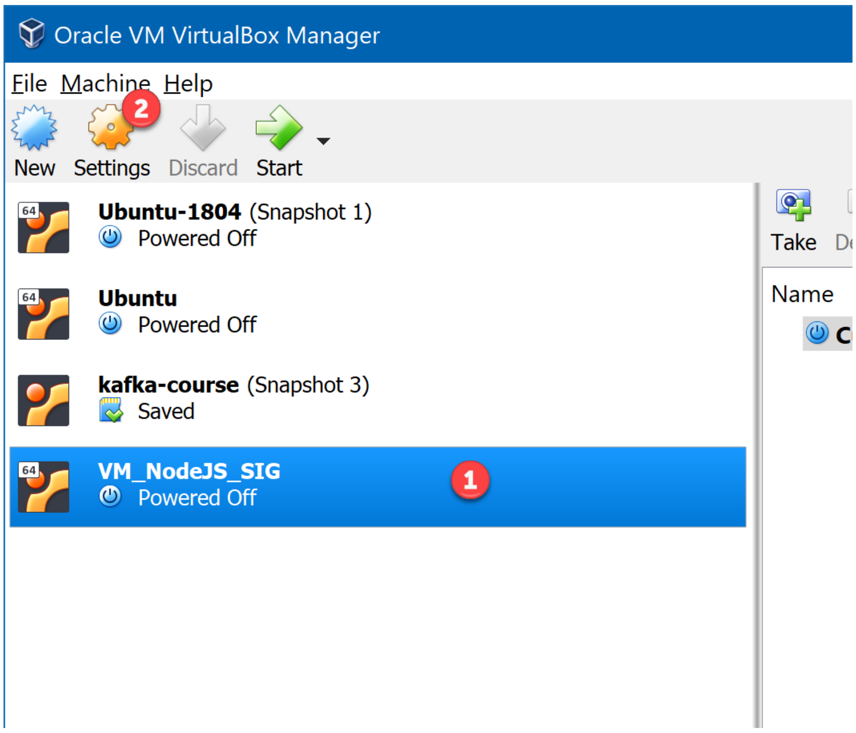
Virtual System 1	
Name	VM_NodeJS_SIG
Description	Deze VM is klaar gemaakt voor het gebr...
Guest OS Type	Ubuntu (64-bit)
CPU	2
RAM	4096 MB
DVD	✓
USB Controller	✓
Sound Card	✓ ICH AC97
Network Adapter	✓ Intel PRO/1000 MT Desktop (82540EM)
Storage Controller (IDE)	PIIX4
Storage Controller (IDE)	PIIX4
Storage Controller (SATA)	AHCI
Virtual Disk Image	C:\Users\Henk Jan van Wijk\VirtualBox V...

✓ Reinitialize the MAC address of all network cards

Appliance is not signed

If you want, you can rename the VM by double clicking on the name.
Select “”**Reinitialize the MAC address of all network cards**” and press Import.

After import start Settings.



Ubuntu-1804 (Snapshot 1) Powered Off

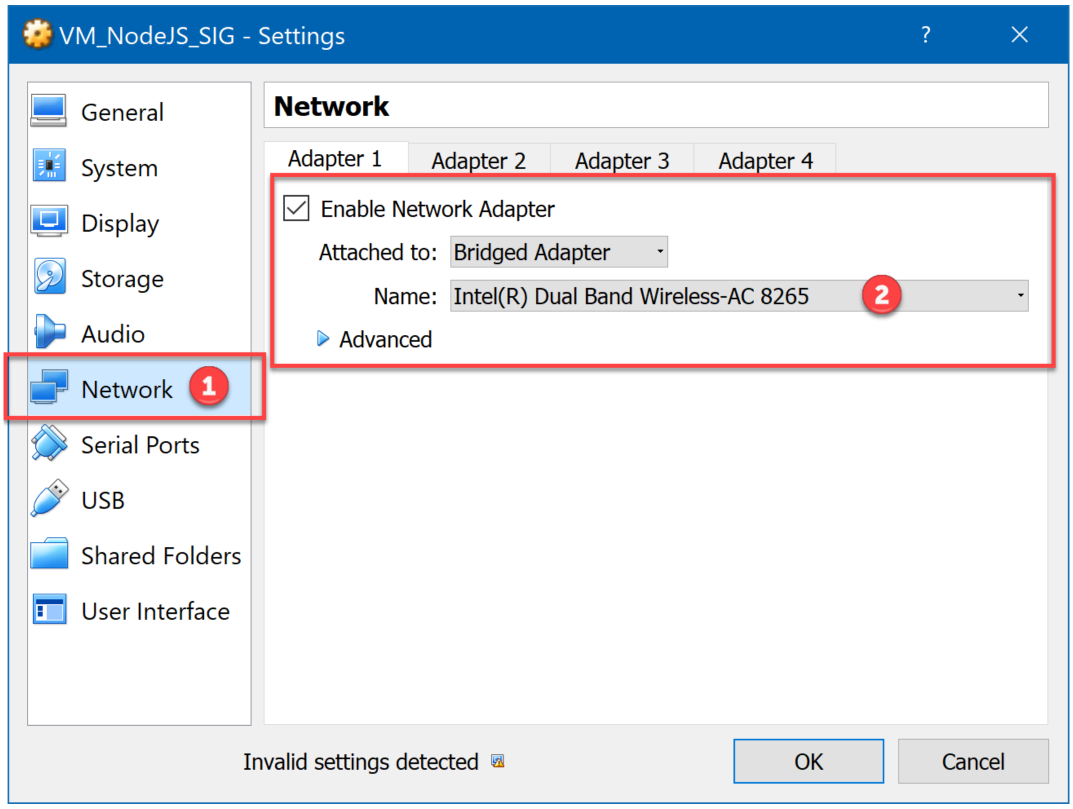
Ubuntu Powered Off

kafka-course (Snapshot 3) Saved

VM_NodeJS_SIG Powered Off

Check the network settings.

Set at least one of the network adapters to NAT so you will be able to connect to the internet from within the VM.



VM_NodeJS_SIG - Settings

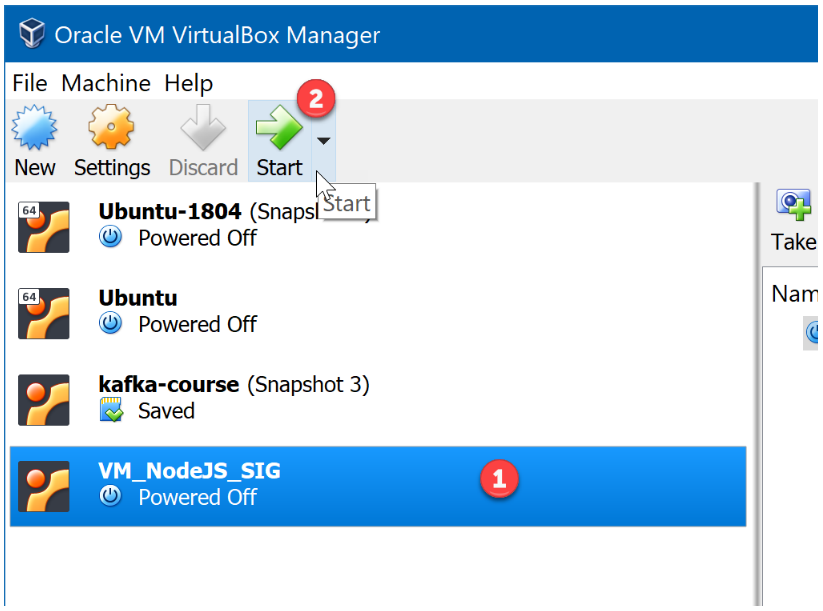
Network

Adapter 1 | Adapter 2 | Adapter 3 | Adapter 4

- [✓] Enable Network Adapter
- Attached to: Bridged Adapter
- Name: Intel(R) Dual Band Wireless-AC 8265
- Advanced

Invalid settings detected

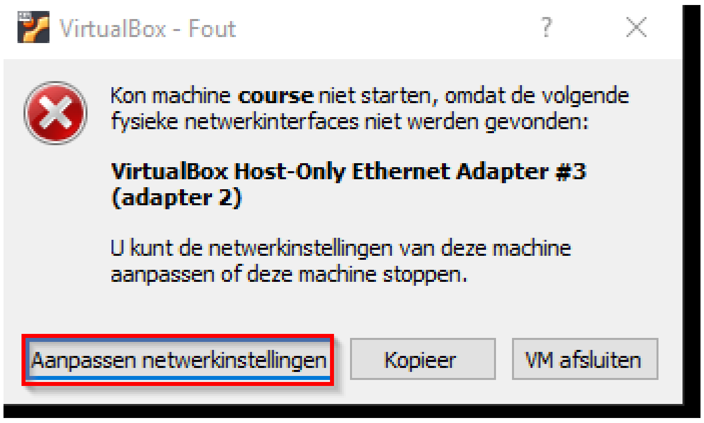
Start the VM



Oracle VM VirtualBox Manager

- **Ubuntu-1804** (Snapsl…) Powered Off
- **Ubuntu** Powered Off
- **kafka-course** (Snapshot 3) Saved
- **VM_NodeJS_SIG** Powered Off

If you get the following error:



VirtualBox - Fout

Kon machine **course** niet starten, omdat de volgende fysieke netwerkinterfaces niet werden gevonden:

VirtualBox Host-Only Ethernet Adapter #3 (adapter 2)

U kunt de netwerkinstellingen van deze machine aanpassen of deze machine stoppen.

the network settings are not set correctly and check them again and make sure you select and appropriate network adapter which exists on your machine.

Introducing the VM

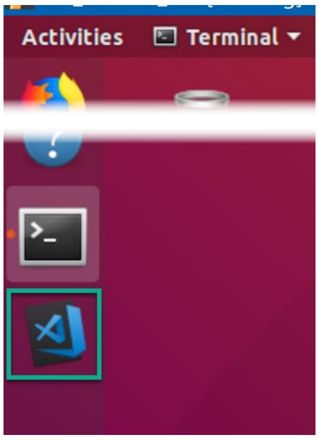
The VM will auto login with user developer and password **Welcome01**.
If at any time there is a request for a password, Welcome01 is the answer.

Set the shared clipboard to bidirectional to be able to copy & paste from and to the VM.
Use `ctrl+shift+v` to paste the clipboard in the VM.

Installed tools

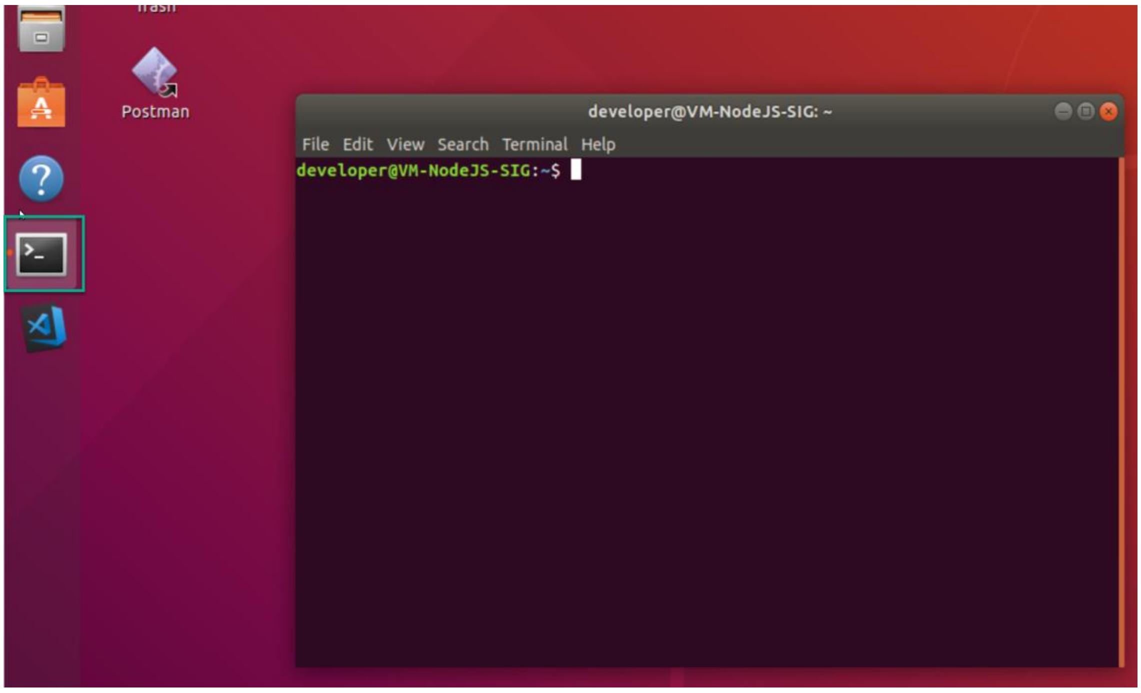
Visual Studio Code

A shortcut to start Visual Studio Code can be found in the Application Bar on the left side of the screen.



Terminal

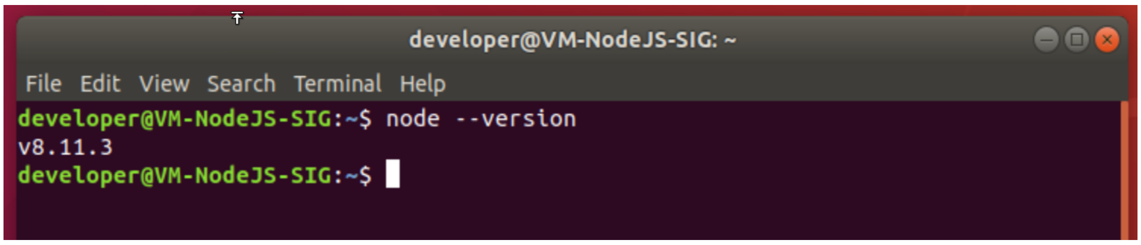
You will need the terminal window several times, a shortcut is also available on the Application Bar.


Postman
developer@VM-NodeJS-SIG: ~
File Edit View Search Terminal Help

```
developer@VM-NodeJS-SIG:~$
```

NodeJS

NodeJS is available from the terminal window.

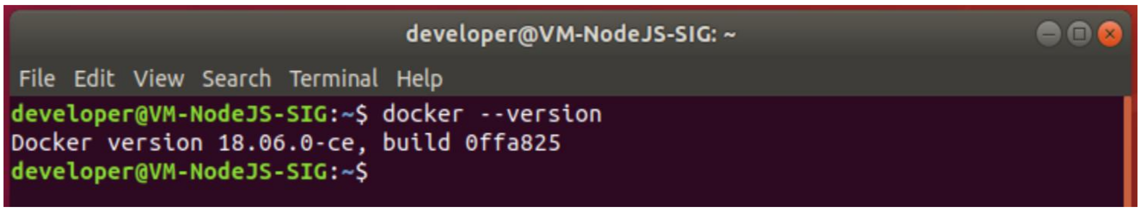

developer@VM-NodeJS-SIG: ~
File Edit View Search Terminal Help

```
developer@VM-NodeJS-SIG:~$ node --version
v8.11.3
developer@VM-NodeJS-SIG:~$ 
```

Type `node --version` to see the version of Node.

Docker

Docker is available from the terminal window.


developer@VM-NodeJS-SIG: ~
File Edit View Search Terminal Help

```
developer@VM-NodeJS-SIG:~$ docker --version
Docker version 18.06.0-ce, build 0ffa825
developer@VM-NodeJS-SIG:~$
```

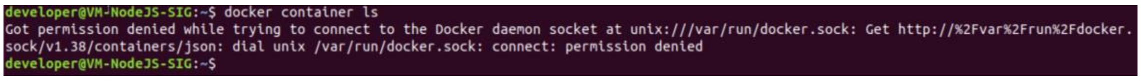
Type `docker --version` to see the version of Docker.

Some useful Docker commands:

`docker ps`
`docker container ls`
	List the running Docker containers. Add `-a` to show also the non-running.

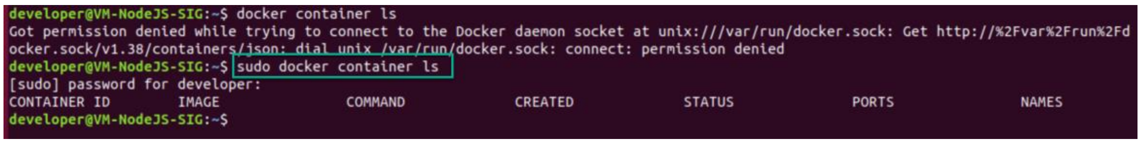
`docker image ls`	List the installed Docker images
`docker run --name` *`<container name>`* *`<image name>`*`:`*`<image tag>`*	Start and run an new Docker container. Add `--rm` to automatically remove the container upon exit / stopping.
`docker start` *`<container name>`*	Start an existing (non-running) Docker container
`docker exec -it` *`<container name>`* `/bin/bash`	Login into a running container using bash shell

If you see the following error when executing a docker command:



```
developer@VM-NodeJS-SIG:~$ docker container ls
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.
sock/v1.38/containers/json: dial unix /var/run/docker.sock: connect: permission denied
developer@VM-NodeJS-SIG:~$
```

Prefix the command with sudo:



```
developer@VM-NodeJS-SIG:~$ docker container ls
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fd
ocker.sock/v1.38/containers/json: dial unix /var/run/docker.sock: connect: permission denied
developer@VM-NodeJS-SIG:~$ sudo docker container ls
[sudo] password for developer:
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
developer@VM-NodeJS-SIG:~$
```

If you do not want to run every docker command using sudo, do the following:

Create the docker group.

```
sudo groupadd docker
```

Add your user to the docker group.

```
sudo usermod -aG docker $USER
```

Log out and log back in so that your group membership is re-evaluated.

Postman

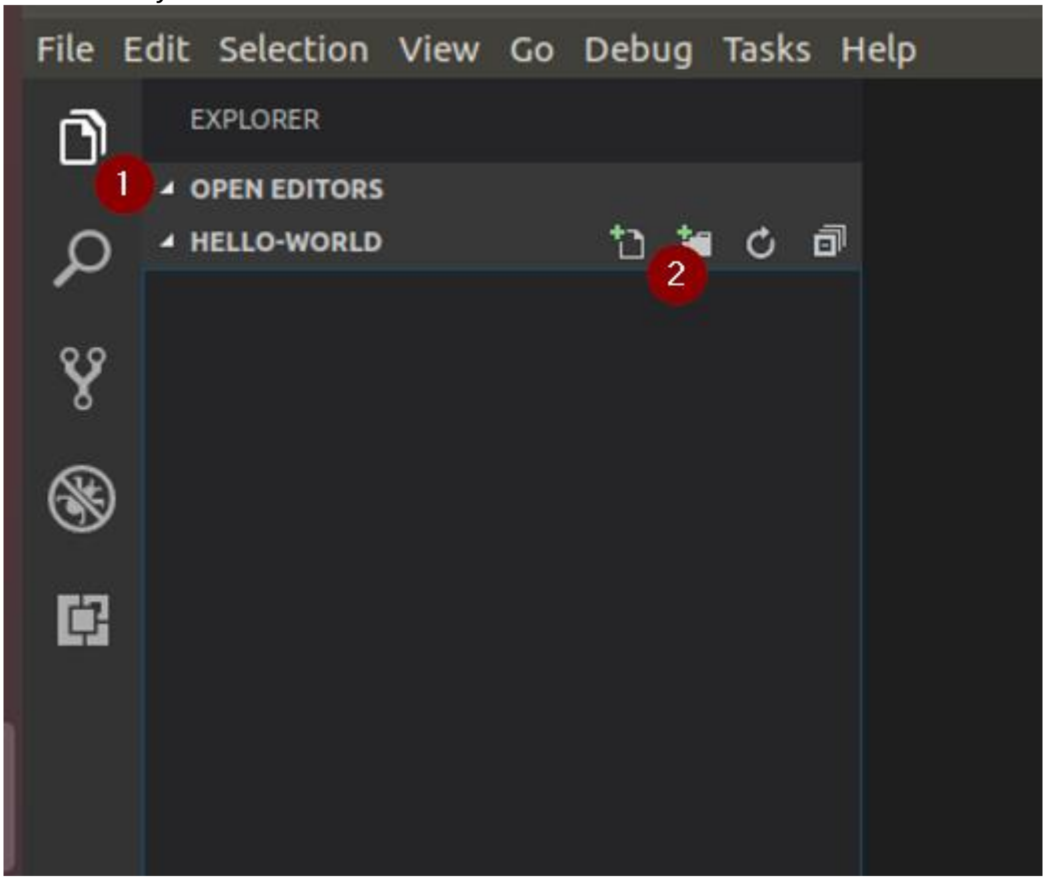
Postman will be used for testing and is available as shortcut on the desktop.

Starting out with NodeJS: Hello world!

Starting a new project

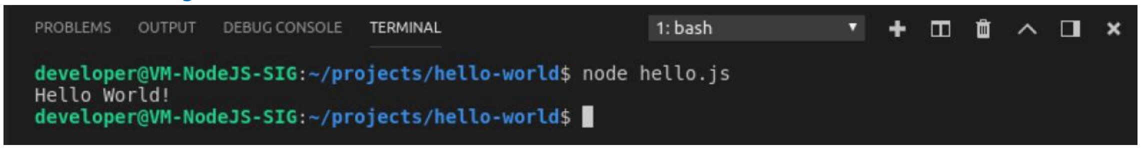
To start a new NodeJS project begin by creating a project folder:

- Open the terminal
- The terminal should be opened in your home directory: ~. You can always go back to the home directory by typing `cd` or `cd ~`
- create a new folder by typing:
  ```
  mkdir projects
  ```
- switch to the projects folder you just made:
  ```
  cd projects
  ```
- create a subdirectory called hello-world:
  ```
  mkdir hello-world
  ```
- switch to the hello-world folder:
  ```
  cd hello-world
  ```
- Open Visual Studio Code (VSC):
  ```
  code .
  ```
- In VSC create a new file and call it **hello.js**
  - Click on the Explorer menu
  - Hover over your folder name and click the leftmost icon



- Type out the code to print the 'Hello World!' statement to the terminal
  ```
  console.log('Hello World!');
  ```

- Save your file.
- Test your code by opening the integrated terminal window (View > Integrated Terminal or CTRL+`
- Type:
  `node hello.js`



```
developer@VM-NodeJS-SIG:~/projects/hello-world$ node hello.js
Hello World!
developer@VM-NodeJS-SIG:~/projects/hello-world$
```

- Instead of just calling the code, you can also wrap it in a function.
- Change the code to (or create a new file if you wish):

```
function say_hello() {
    console.log('Hello World!');
}

say_hello();
```

- And run the code again. You should see the same result.
- Change the code to include a function parameter and run het again:

```
function say_hello(name) {
    console.log('Hello ' + name + '!');
}

say_hello('me');
```

- Or if you wish to use a command line argument:

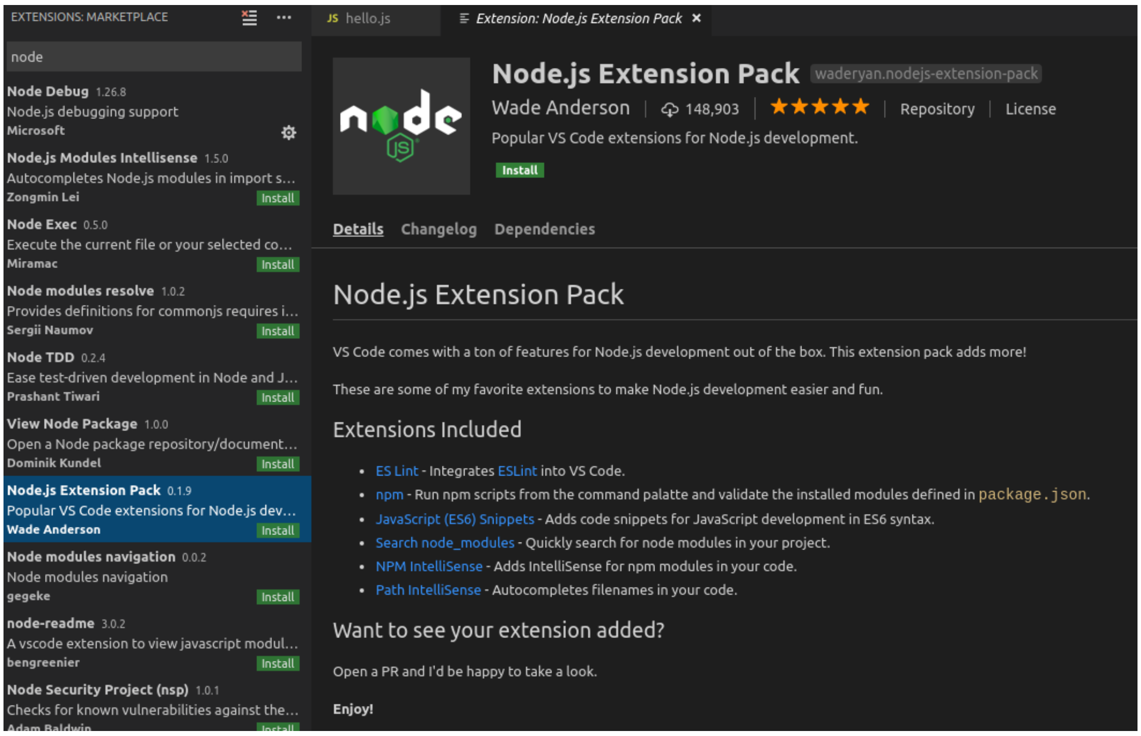
```
function say_hello(greet) {
    console.log('Hello ' + name + '!');
}

// assume command line: node hello.js someName
var name = process.argv[2];
say_hello(name);
```

- Run by typing `node hello.js yourname` or if you want to have spaces: `node hello.js “your name”`

Now we will add some extensions to Visual Studio Code (VSC).

- Go to the extensions menu in VSC.
  - CTRL+SHIFT+X or the icon in the left bar
- Find the Node.js Extension Pack and click install, then restart VSC



Node Debug 1.26.8
Node.js debugging support
Microsoft

Node.js Modules Intellisense 1.5.0
Autocompletes Node.js modules in import s…
Zongmin Lei
Install

Node Exec 0.5.0
Execute the current file or your selected co…
Miramac
Install

Node modules resolve 1.0.2
Provides definitions for commonjs requires i…
Sergii Naumov
Install

Node TDD 0.2.4
Ease test-driven development in Node and J…
Prashant Tiwari
Install

View Node Package 1.0.0
Open a Node package repository/document…
Dominik Kundel
Install

Node.js Extension Pack 0.1.9
Popular VS Code extensions for Node.js dev…
Wade Anderson
Install

Node modules navigation 0.0.2
Node modules navigation
gegeke
Install

node-readme 3.0.2
A vscode extension to view javascript modul…
bengreenier
Install

Node Security Project (nsp) 1.0.1
Checks for known vulnerabilities against the…
Adam Baldwin

Node.js Extension Pack waderyan.nodejs-extension-pack
Wade Anderson | 148,903 | ★★★★★ | Repository | License
Popular VS Code extensions for Node.js development.
Install

Details Changelog Dependencies

Node.js Extension Pack

VS Code comes with a ton of features for Node.js development out of the box. This extension pack adds more!

These are some of my favorite extensions to make Node.js development easier and fun.

Extensions Included

- ES Lint - Integrates ESLint into VS Code.
- npm - Run npm scripts from the command palatte and validate the installed modules defined in package.json.
- JavaScript (ES6) Snippets - Adds code snippets for JavaScript development in ES6 syntax.
- Search node_modules - Quickly search for node modules in your project.
- NPM IntelliSense - Adds IntelliSense for npm modules in your code.
- Path IntelliSense - Autocompletes filenames in your code.

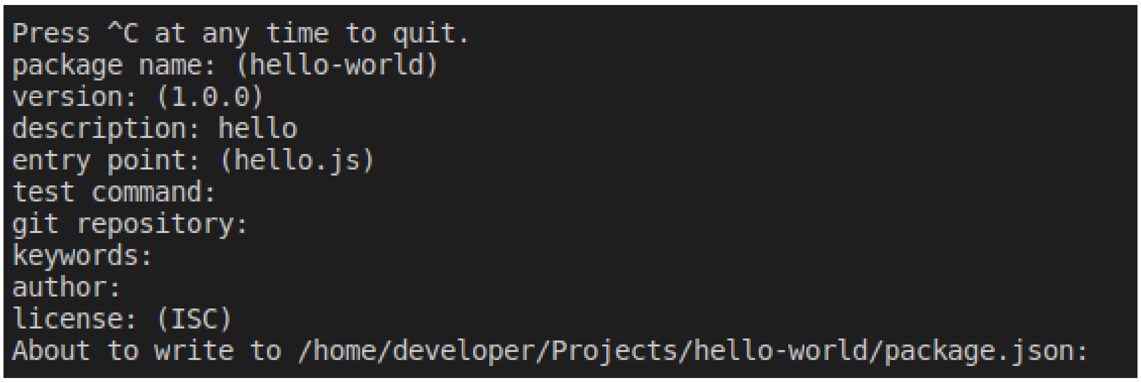
Want to see your extension added?

Open a PR and I'd be happy to take a look.

Enjoy!

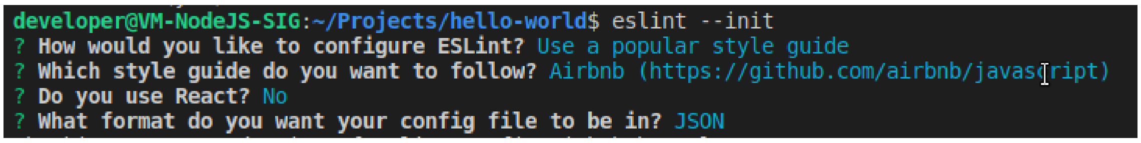
- ○

- Next we need to install ESLint by typing the command in the terminal (in VSC) and restart VSC again
  `sudo npm install -g eslint`
- Now create a package.json file and use the settings from the picture. The package.json file contains the settings for your new project.
  `npm init`



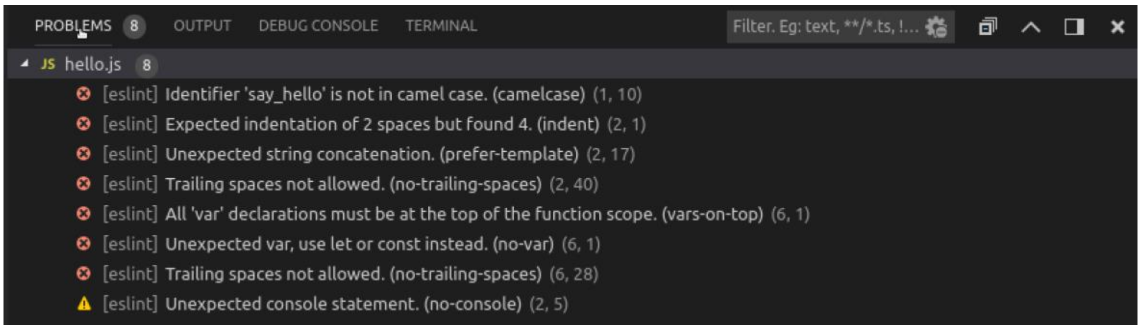
```
Press ^C at any time to quit.
package name: (hello-world)
version: (1.0.0)
description: hello
entry point: (hello.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /home/developer/Projects/hello-world/package.json:
```

- Next execute the following command to set up ESLint. ESLint uses style guides to ensure a uniform coding style.
  `eslint --init`



```
developer@VM-NodeJS-SIG:~/Projects/hello-world$ eslint --init
? How would you like to configure ESLint? Use a popular style guide
? Which style guide do you want to follow? Airbnb (https://github.com/airbnb/javascript)
? Do you use React? No
? What format do you want your config file to be in? JSON
```

- Install the npm dependencies when asked
- Restart VSC for one last time.
- ESLint will now give out errors and a warning in the Problem tab:



```
[eslint] Identifier 'say_hello' is not in camel case. (camelcase) (1, 10)
[eslint] Expected indentation of 2 spaces but found 4. (indent) (2, 1)
[eslint] Unexpected string concatenation. (prefer-template) (2, 17)
[eslint] Trailing spaces not allowed. (no-trailing-spaces) (2, 40)
[eslint] All 'var' declarations must be at the top of the function scope. (vars-on-top) (6, 1)
[eslint] Unexpected var, use let or const instead. (no-var) (6, 1)
[eslint] Trailing spaces not allowed. (no-trailing-spaces) (6, 28)
[eslint] Unexpected console statement. (no-console) (2, 5)
```

- First have a look at the warning. ESLint throws this warning because it is unusual to have a console.log command in a front-end application. But because this is a console based application we do not want to have that check in place. The warning can be turned off by editing the .eslintrc.json file like this:

```
{
    "extends": "airbnb-base",
    "rules":{
        "no-console": "off"
    }
}
```

- When you save the file, you will notice that warning will disappear.
- Now check the errors and try to fix them for yourself.
- Hint 1: Use F2 or right click > Rename symbol to refactor
- Hint 2: press CTRL + SHIFT + P to open the VSC command palette. Here you can execute all kinds of commands (builtin or added by extensions). Search for eslint.

If you want to know more about the used style guide, go to: https://github.com/airbnb/javascript.

Using server-side NodeJS

Using NodeJS to read a connection file

- First let's start a new project by opening or navigate to the Linux terminal again and type:

  ```
  cd ~/projects
  mkdir read-file
  cd read-file
  code .
  ```
- A new Visual Studio Code window is opened (you can close the other one if it is still open).
- Reopen the command window
- execute the `npm init` command, accept the default settings except the entry point. Name it: **read-file.js**
- make a config file called **config.json**. This contains the connection settings to mongoDB:

```
{
    "host": "localhost",
    "port": 27017,
    "db": "catalog"
}
```

- We will use NodeJS to read the connection settings from this file. Start by creating a new file called **read-file.js**

```
 const fs = require('fs');

let config;

function readConfiguration() {
  fs.readFile('config.json', (err, data) => {
    if (err) {
      throw err;
    }
    config = data;
    console.log(config);
  });
}

readConfiguration();
```

- If you execute this file (in the VSC terminal: `node read-file.js`), it will show you the buffer object, into which the connection data is loaded.
- Change the line `config = data` to `config = JSON.parse(data);` to convert the buffer to JSON. If you run the file again you will see the content of the file in a readable form.

  Let's explain the code so you understand what is happening:

  ```
  const fs = require('fs');
  ```

This statement declares a constant fs with the reference to the fs (file system) module. require itself is also a (core) module of node.js. You can use the fs constant in the rest of your file to call functions in that module.

`let config;`
Define a new variable named config in the global scope (it is not defined in a block). Note that there is no typing. config can be any type. If you want you can assign a value directly to initialize it.

`function readConfiguration()`
The function who read the file.

`fs.readFile('config.json', (err, data) => {…}`
Call the readFile function in the fs module and register a callback function with 2 arguments: err and data.
The => is called an (fat) arrow function which is a shorter syntax for: `function` *`<function name> (arguments)`* `{}`
The callback function is called when the file has been read. Note that the application will continue. If you add for instance the following statement after the call to readConfiguration(): `console.log('Done.');`
You will notice that in the output you will see `Done` first.

Arrow function

The arrow function was introduced with version ES6.

Example code in ES5 syntax:

```
function timesTwo(params) {
  return params * 2
}
```

Some code in ES6 syntax:

```
let timesTwo = params => params * 2
```

Use-case

Description

We will use a very simple use case to demonstrate how to use node.js for database and API use.

Consider we have to product catalog containing computer (or parts) we sell. And we have several sales persons who are the main contact for certain product groups.
We have 2 tables: one with the products and one with all the sales persons and the product groups they sell. In the end we need to support an application which can search in de product catalog and give the product details plus the salesperson which is responsible for the found products.

The products 'table' will consist of the following fields:

- productgroup (string), e.g. "Laptops"
- brand (string), e.g. "HP"
- name (string), e.g. "Elitebook 850 G5"
- modeltype (string), e.g. "i7-8550U"
- modelnumber, e.g. "3JX19EA#ABH"

The salespersons 'table' will consist of the following fields:

- name (string), e.g. "Dijkstra"
- initials (string), e.g. "J"
- prefix (string), e.g. ""
- productgroups (list), e.g. "Laptops, Desktops"

Setup

For implementing the use case we need to have access to a mongodb database. If added the developer user to the docker group you won't need to use sudo.

- Open the terminal
- Create a data directory:
  ```
  mkdir ~/data
  mkdir ~/data/mongo
  ```
- and run this command: `sudo docker run -d --name mongodb -p 27017:27017 -v ~/data/mongo:/data/db mongo:latest`
- Check if the container is running: `sudo docker exec -it mongodb mongo`
- Type `exit` to exit the mongo shell

Create a new project:

```
cd ~/projects
mkdir catalog
cd catalog
code .
```

In Visual Studio Code open the integrated terminal and enter:

```
npm init
eslint –init
```

just like we have done before. Use as entry file: start.js
Then add the required module for mongo and working with rest api's (-save adds module this to the package dependencies):

```
npm install mongodb -save
npm install mongoose -save
npm install express -save
npm install body-parser -save
```

Adding data

Now add the file **setupdb.js** to add some data to our mongodb database.

We will create a **catalog** database with a **products** collection. An add some sample data to start with.

We can do this using just the mongodb module as we see first. After that we will refactor the code to use mongoose, a ORM module for mongo.

Tip: to check if data is actually added to your database: open a Linux terminal window and type: `docker exec -it mongodb mongo`
You are now in the mongo shell.
Type: `use catalog` *to switch to the catalog database.*
Type: `db.products.find({})` *to query alle documents in the products collection.*

Step 1:

```
const mongo = require('mongodb');

const mongoClient = mongo.MongoClient;
const url = 'mongodb://localhost:27017/catalog';

mongoClient.connect(url, (err, db) => {
  if (err) throw err;
  console.log('Connected to mongodb.');
  db.close();
});
```

Try and run the code. If all goes well you should see the message.
You will get a warning. Change the code according to the given hint:

```
mongoClient.connect(url, { useNewUrlParser: true }, (err, db) => {
```

And try again. The warning should be gone.

Change the code to:

```
const mongo = require('mongodb');

const mongoClient = mongo.MongoClient;
const url = 'mongodb://localhost:27017/catalog';

mongoClient.connect(url, { useNewUrlParser: true }, (err, db) => {
  if (err) throw err;
  console.log('Connected to mongodb.');
  const dbo = db.db('catalog');
  dbo.createCollection('products', (errCollection, res) => {
    if (errCollection) throw errCollection;
    console.log('Collection products created.');
    db.close();
  });
});
```

Run the code (`node setupdb.js`) to check if you get two messages.
A new database with a collection has been created.

To insert a new record into the collection, add to following code after console.log('Collection products created.'):

```
      products.insertOne(product, (errInsert, res) => {
      if (errInsert) throw errInsert;
      console.log('1 document inserted');
      db.close();
    });
```

Add a variable named products and one named product (after dbo):

```
  const products = dbo.collection('products');
  const product = {
    brand: 'HP',
    productgroup: 'laptops',
    name: 'EliteBook 1040 G4',
    modeltype: 'i5-7200U',
    modelnumber: '1EP75EA#ABH',
  };
```

And remove the db.close() statement in the outer block. We want to close the database connection after inserting the record.
If you leave the db.close() statement, it will be executed before the insert!
This method of adding data could be made better of course. You could move the insert into a separate function. But you need to keep in mind the asynchronous nature.
Let's say you add a function addProduct:

```
function addProduct(coll, product) {
  coll.insertOne(product, (errInsert, res) => {
    if (errInsert) throw errInsert;
    console.log('1 document inserted');
  });
}
```

Then you can call this function to add the product:

```
  dbo.createCollection('products', (errCollection, res) => {
    if (errCollection) throw errCollection;
    console.log('Collection products created.');

    addProduct(products, product);
  });
```

But now you do not close the database connection anymore. Where would you put it?

***Assignment*:** try to figure it out for yourself (*hint: you need a callback function*).

Introducing mongoose

One way to make it easier is to use a ORM (Object Relational Mapper) module or in this case a ODM (Object Document Mapper) because mongodb is a nosql document database, no relational database. For mongo we will use the mongoose module (which we installed already). More information can be found on https://mongoosejs.com/.

For each of our 'tables' (products and salespersons) we will create a schema/model which can be used to easily work with the data.

Create a new file named products.js with the following content:

```
const mongoose = require('mongoose');

// Declare Mongoose schema for products
// Unique: productgroup + brand + modelnumber
const productSchema = new mongoose.Schema({
  productgroup: { type: String, required: true, unique: true },
  brand: { type: String, required: true, unique: true },
  name: { type: String, required: true },
  modeltype: { type: String, required: true },
  modelnumber: { type: String, required: true, unique: true },
});

// Declare Mongoose model to used as class
const Product = mongoose.model('Product', productSchema);

module.exports.Product = Product;
```

rename setupdb.js to setupdb_0.js and create a new file **setupdb.js** which will use mongoose instead of directly calling mongodb functions:

```
const mongoose = require('mongoose');
const products = require('./products');
const url = 'mongodb://localhost:27017/catalog';

mongoose.connect(url, { useNewUrlParser: true });
const db = mongoose.connection;

db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', () => {
  // we're connected!
  const product = new products.Product({
    brand: 'HP',
    productgroup: 'laptops',
    name: 'EliteBook 850 G5',
    modeltype: 'i7-8550U',
    modelnumber: '3JX19EA#ABH',
  });
  product.save((err, record) => {
    if (err) return console.error(err);
    console.log(`Added: ${record.brand} ${record.name}`);
    db.close();
    return true;
  });
});
```

And run the code: `node setupdb.js`
And check if the record is added.
Try to run it again to see if the same record is added or not? Why?

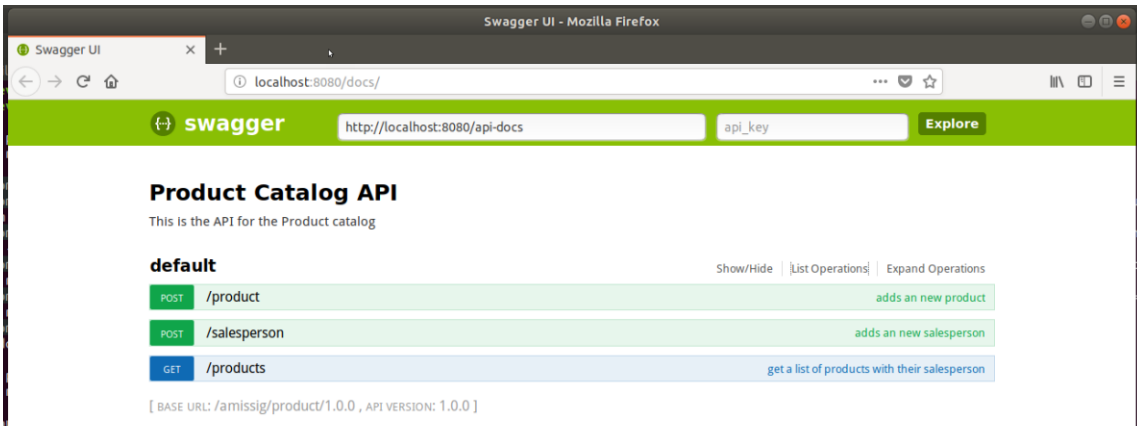
Now create a file called salespersons.js for the Salespersons collection.
And update the setupdb.js to create add a salesperson.

Swagger definition

Go to https://github.com/AMIS-Services/sig-nodejs-integration-dev for the complete description of the API.
You can also clone the repo:
Open a Linux terminal window and type:

```
cd ~/projects/
git clone https://github.com/AMIS-Services/sig-nodejs-integration-
dev.git
cd sig-nodejs-integration-dev/productapi
npm start
```

Open a web browser with url: http://localhost:8080/docs/
You can use this front-end application to get an understanding of how our back-end application should work.


Swagger UI - Mozilla Firefox

localhost:8080/docs/

http://localhost:8080/api-docs

Product Catalog API

This is the API for the Product catalog

default

POST /product adds an new product

POST /salesperson adds an new salesperson

GET /products get a list of products with their salesperson

[BASE URL: /amissig/product/1.0.0 , API VERSION: 1.0.0]

The swagger file can be found in the api subfolder.

REST API

In the previous paragraph we added the record directly using mongoose. Now we will add a REST API to add new records to the products and salespersons collection and to query the data.

For the inserts we will create two REST API methods:

1. POST method on /product
2. POST method on /salesperson

Both POST methods will have the complete document for each of the collections as body.

For query we will create one REST API method:

1. GET method on /products

Create a new folder, we called it sig-nodejs

```
cd ~/projects
mkdir sig-nodejs
cd sig-nodejs
code .
```

Start by making a new file called product-app.js.
Initiate npm and eslint. Install mongoose, express and body-parser.
Open the product-app.js file.

```
const express = require('express');
const app = express();

module.exports = app;
```

This module will be used to configure the app we will be creating.

Now create a new file called product-api-server.js

```
const app = require('./product-app');

const port = process.env.PORT || 3000;
const server = app.listen(port, () => {
  console.log(`Express server listening on port ${port}`);
});
```

This is the the server for the app. Once this file is ran with

```
node product-api-server.js
```

the server will be listening on port 3000. It can be reached there with Postman at the address: http://localhost:3000/

Now create a file called product-api-db.js:

```
const mongoose = require('mongoose');
const fs = require('fs');

function connectToDB(connection) {
  mongoose.connect(`mongodb://${connection}`, { useNewUrlParser:
true });
}

function readConfiguration(callback) {
  fs.readFile('config.json', (err, data) => {
    if (err) {
      throw err;
    }
    const config = JSON.parse(data.toString('utf8'));
    const {
      host,
      port,
      db,
    } = config;
    console.log(config);
    const connection = `${host}:${port}/${db}`;
    callback(connection);
  });
}

readConfiguration(connectToDB);
```

this file is used for connecting to the database.

The next step is to again define the product schema. Create a file called product-api-products.js, and use the following code:

```
const mongoose = require('mongoose');

const productSchema = mongoose.Schema({
  type: { type: String },
  manufacturer: { type: String },
  model: { type: String },
  modelno: { type: String },
});
productSchema.index({ manufacturer: 1, model: 1, modelno: 1 }, {
unique: true });

const Product = mongoose.model('Product', productSchema);

module.exports = Product;
```

The productSchema.index ensures that the combination of manufacturer, model and modelno are unique.

Now that we have an outline for an application, we need to tell it that the mongo database is accessible. We can do this by adding the code:

```
const db = require('./product-api-db');
```

to the product-app.js file. Now the application knows mongoDB is open for connections.

Now comes the trickiest part. We need to create a controller for our application. This will control (hence the name) the flow of our application. We shall call it product-api-productcontroller.js, and enter the following code:

```
const express = require('express');
const bodyParser = require('body-parser');
const Product = require('./product-api-products');

const router = express.Router();
router.use(bodyParser.urlencoded({ extended: true }));
router.use(bodyParser.json());
```

This is the basic layout of the controller. This will tell the application what to do.
To enable the controller to work we need to tell the application to actually use the controller. We can do this by changing the application code slightly. In the product-app.js file paste-over the following code:

```
const express = require('express');
const db = require('./product-api-db');
const ProductController = require('./product-api-
productcontroller');

const app = express();
app.use('/product', ProductController);

module.exports = app;
```

We can see that the ProductController is added. And this time the app is actually used (app.use line). after the app.use we can see that it is listening to the /product endpoint, which makes the full address: http://localhost:3000/product.

We can add the following code to the controller:

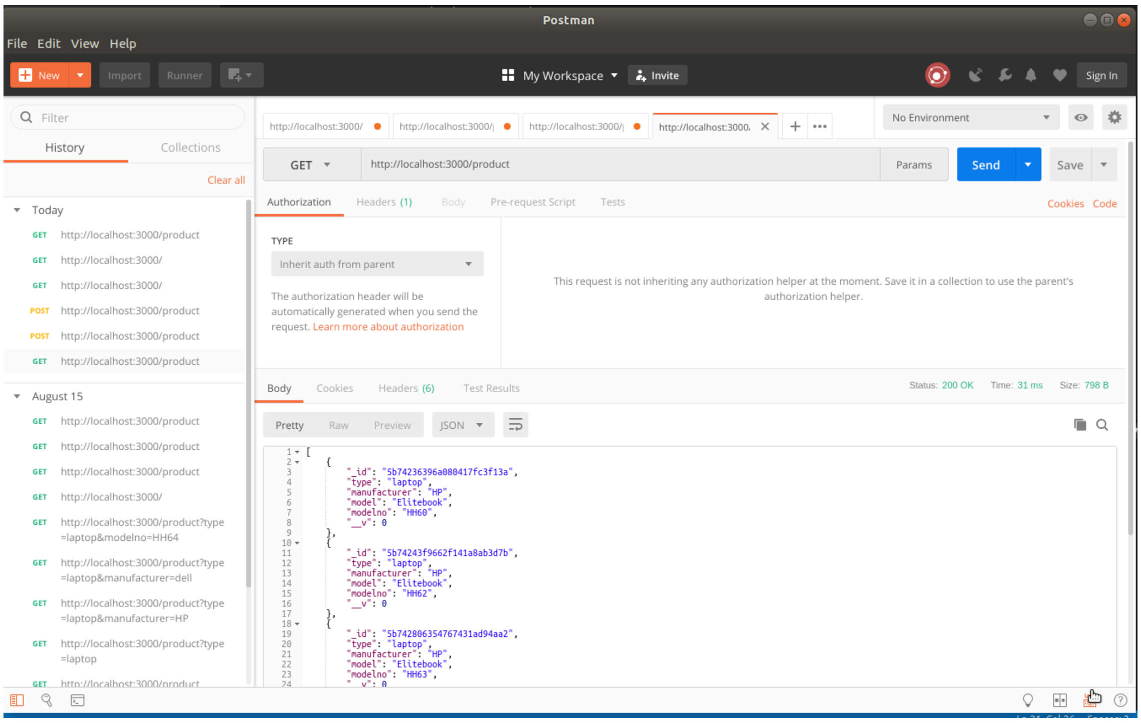
```

router.get('/', (req, res) => {
  Product.find({}, (err, products) => {
    if (err) return res.status(500).send('{"message": "Could not
find product"}');
    res.status(200).send(products);
    return products;
  });
});
```

This code does the following:
it will trigger once a get is done at the http://localhost:3000/product address. It will then execute the db.products.find({}) command on mongo db. If this returns an error it will return the error message. If it finds products it will return them as a json file.

Now run the application by typing node product-api-server.js into the terminal in VSC. It should say it is listening on port 3000 and also return the configuration of the database.

Now in postman we can create a new “get”



Today
GET http://localhost:3000/product
GET http://localhost:3000/
GET http://localhost:3000/
POST http://localhost:3000/product
POST http://localhost:3000/product
GET http://localhost:3000/product

August 15
GET http://localhost:3000/product
GET http://localhost:3000/product
GET http://localhost:3000/product
GET http://localhost:3000/
GET http://localhost:3000/product?type=laptop&modelno=HH64
GET http://localhost:3000/product?type=laptop&manufacturer=dell
GET http://localhost:3000/product?type=laptop&manufacturer=HP
GET http://localhost:3000/product?type=laptop
GET http://localhost:3000/product

GET http://localhost:3000/product

TYPE
Inherit auth from parent

The authorization header will be automatically generated when you send the request. Learn more about authorization

This request is not inheriting any authorization helper at the moment. Save it in a collection to use the parent's authorization helper.

Status: 200 OK Time: 31 ms Size: 798 B

```
[
    {
        "_id": "5b74236396a080417fc3f13a",
        "type": "laptop",
        "manufacturer": "HP",
        "model": "Elitebook",
        "modelno": "HH60",
        "__v": 0
    },
    {
        "_id": "5b74243f9662f141a8ab3d7b",
        "type": "laptop",
        "manufacturer": "HP",
        "model": "Elitebook",
        "modelno": "HH62",
        "__v": 0
    },
    {
        "_id": "5b742806354767431ad94aa2",
        "type": "laptop",
        "manufacturer": "HP",
        "model": "Elitebook",
        "modelno": "HH63",
        "__v": 0
```

Now comes the trickier part. We will also need to be able to add new products via postman. You will mostly need to do this by yourself. Here are a few tips to help get you started:

Tip 1: Since this a new branch in the flow, you will need to alter the controller.
Tip 2: Use a post in postman to send data to your app.
Tip 3: We have already made an option to save data to mongoDB, you can partially reuse code.
Tip 4: We can extract data from the request message by using the req.body.#varname# statement.