40)Module Introduction

Lets see how e can make development easier and how we can speed it up a little bit and most importantly, how we can actually debug our nodejs apps or how wecan use varius tools to understand better what’s going on in our nodejs application. In this module you will learn some tricks and trips to which will hopefully help you write nodejs applications in a very efficient manner.

41)Understading npm scripts

In order to run our file fater every chnag we have to run this-

**node app.js**

but it actually is possible to define some scripts in nodejs project that can help us with tasks like this, we can also us ethem for other tasks but especially for this, they can be useful right now and for that we have to use a feature we did’nt use thus far. We have to use npm. we can use it to download third party libraries but w ecan also use npm to initialize so called node project or to add some extra features to it to be precise. Because we already got a node project here but now in this node project, in terminal(after navigating to project folder) run-

**npm init**

you will be asked coiple of questions.

First it will ask for name of package, for now yon simply use default option which is name of project.in () you will get suggestion, default name. to select it just press enter.

Then it will ask for description, entry point(for now this entry point will not do anything)

Then it will ask for test command, you can leave it empty. Then al last it will ask for licence, but if you dnt plan on sharing your code publically , then this does not matter.

What you will get is package.json file-

{

"name": "nodejs",

"version": "1.0.0",

"description": "",

"main": "app.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "Sumeet Sood",

"license": "ISC"

}

Here you can see all these proeprties and you can edit them if you want. It is in json format, which looks a lot like js objects except for the fact that here properties are always between “” so are values except for numbers or arrays or true or flase, which are not put between these. This package.json is configuration file for your project. Now what does this configuration file give you? We can see that w have scripts section there. It has one script which will not do anything. But we can add our own scripts here. Lets see how-

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

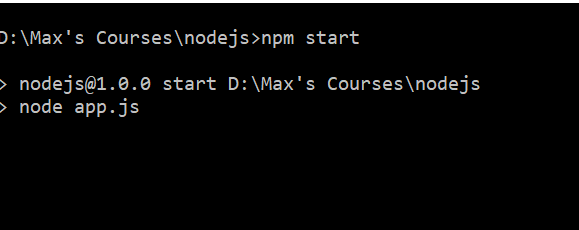
"start": "node app.js"

},

Now start here is special script name as you will see in a second. so to make sure to type it correctly.

When you run-

**npm start**, it will run your script that you have defined in package.json, in this case **node app.js**



Here we can see that it is running node app.js command.

Now it is not saving us lot of characters but some atleast and its also good practice to because if you ever share this project, it’s pretty common that people just have to run **npm start** and that they wnt have to guess which of your js files is the entry file.

Now as I mentioned that start is a special script name. you can add more scripts without using a special name. you can use anyname you want, just make sure that to alays wrap the name in **“”** and that it does not contain any blanks or whitespaces. Lets add one new script-

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"start": "node app.js",

"start-server" : "node app.js"

},

Ie we now run **npm start-server** , we get an error that this is not a known command. This is because just typing script name after npm wnt work, it worked in case of start because as I told start is a special case. You can run your won scripts with custom names by-

**npm run start-server**

now if you have worked with angular,react or any modern web frontend development workflow, you will have seen that use these scripts a lot ot trigger build workflows for your projects for example and indeed you can use that for all kinds of tasks you want to execute. For now we wnt dive deeper into that. We will see it later when we explore node’d functionality as a build tool.

42)Instaling 3rd party Packages

You can install npm package like this-

**npm install “pakage name”**

this command will install it but dnt hit enter yet. You can define how it hould be installed because packages which you install can be divided into development dependencies and production dependencies. For ex **nodemon** would be development dependency because we only use it during the development process.once we install our app on real server we dnt need it there. And you can basically tell npm which kind of dependency this is, this does not make make a huge difference and you can omit this setting but it helps you understand which package is sued for what. So run-

**npm install nodemon --save-dev**

if you would have ran this-

**npm install nodemon –save**

then this will be production dependency.

Now we will install nodemon. It will automatically recompile and run our code , whenever we change something in our code and save it. It is dev dependency. So run this

**npm install nodemon --save-dev**

it gives us couple of new things in our project, it gives us that node\_modules folder, the package.json file and it updated the package.json file. in package.json, new dev dependency section is added-

"devDependencies": {

"nodemon": "^1.18.5"

}

It has version number of package. Now w ehave character before version. It defines how this package will be updated, if you rerun just **npm install** without defining an extra package name because this command standalone will simply go through all your packages mentioned in package.json and install them and it would automatically pick later version if avalaible. But more on npm and modules can be found in separate module, later in this course.

Now question is where is this package installed? Well that is node\_modules folder. if you see this folder, it a huge folder. the reason for this is , we got nodemon in there. Now this is basically source code of package or build version of package we installed and this package simply happens to have couple of peer dependencies(you can see them in package.json file which is inside nodemon folder in package.json). so these dependnecies are also installed when we install nodemon. Tis is why we see lot of folders in node\_mon folder. you can delete node-modules folder if you want to free up space. But you need to run, **npm install** before you can run this project. It will install all packes mentioned in package.josn file.

Now package-lock.json file by the way stores the exact versions I installed today so if you share you project with others, they get these exact versions too, instead of latest versions. But again more on npm in a separate module.

I have asked a question-

<https://www.udemy.com/nodejs-the-complete-guide/learn/v4/questions/5512496>

44)Using nodemon for autorestarts

We change our start script-

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"start": "nodemon app.js",

"start-server": "node app.js"

},

Now it will look for nodemon module, which it will find in this project because we installed it. If you manually run-

**nodemon app.js**

you will get error because nodemon is only installed in this project and not globally on your machine but terminal will try to find it globally. Here(in scripts tag) it will work because it will look locally(npm scripts will look for the local versions of the package).

46)Understanding different types of errors

Syntax errors- these are like you missed the closing curly braces.

Runtime errors- here you try to execute some code that will just break.

Logical errors - these are most difficult ones. Because here you will v=never see an error message. You app just does’nt work the way it should and it can be hard to find these but I will also show you some tool which can help you with that.

47)Finding and fixing syntax error

Ide will give you error. Like it is syntax error-

Cont url = ‘hj’;

Here you will get error(in console) like unexpected identifier. It will also point out the line.

Another syntax error will be if you forget the closing parenthesis.in this case iDE will show error at the end. Error will be ‘} expected

48)Dealing with runtime error

Example of run time error will be like, we are still adding headers to response object after we have returned it. We will get error only when that handler is executed. In routes.js we are returning the response(in handler of url = ‘/’), to prevent execution of next lines.

Error message would be – cannot set headers after they are sent to client.

49)Logical Errors

These are most difficult ones to fix because we will not get error message, but our app will not behave in a way it should behave. We are getting string entered by user from body here-

const message = parsedBody.split('=')[1];

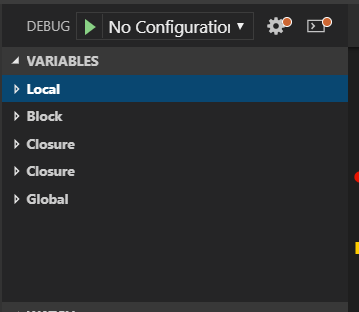
lets say we use index 0 insteda of 1. So here we wnt get any error message. For this type of errors, we can use node debugger, which has great integration into vs code. Lets see how to use it.

Select app.js(open it), this is important.Go to debug -> start debugging.

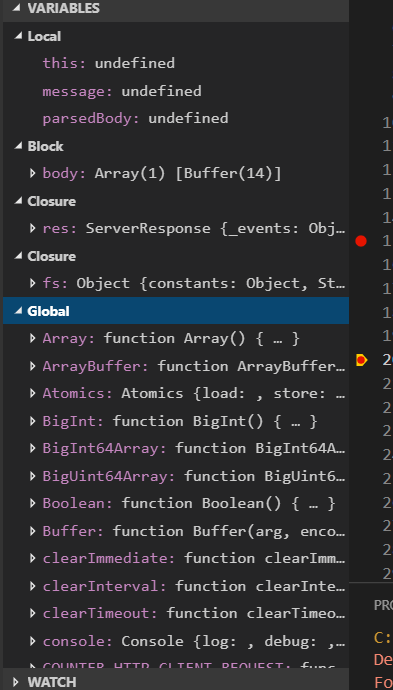
Then you have to choose environment, choose nodejs. Now our code is automatically executed by vs code debugger. In terminal you have debug console where you can see that debug is attached and listening. This means now you can look into your code as it is executing.

To really do that you need to set breakpoints. Now your code will paused here. You can hover over variables to see their value. you can even expand objects, while you hover over variables that store them.

On left hand there is a icon for debugging(or you can also do it by view ->debug). Now you will be in debug mode. You can see menu on left hand side.



You can see variable section. there you can see values of local variables, block variables, variable in closure, variable in block and so on.



We can also have watch block. Where we can check value of various javascript expressions. We can also check value of variable there.Then we have call stack which shows the call stack i.e how we reached this line and so on.lets say we wan to see which function called the function in which we have breakpint. Then call stack is helpful. Click on lement of stack and it will take you to function. Then we have breakpoint section. there we can see which all breakpoints we have. We can disable breakpoints from there and later we can enable them from here, manually setting them from here.

We also have callstack section, which shows how process went through your code and you can click on different parts to see where actually this code which belongs to code that was executed, can be found. And not all of that code is code that you wrote, a lot of that is core nodejs code.

One more thing while you are in debugger you can execute the statement on debug console(it is visual studio code, at bottom along with consodle of visual studio code) and see there output. lets say you have string in a variable, you can do various operations on it and see results. It wnt affect your main code which is running. Same thing can be done on watchlist.

51)Restarting the debugger automatically after editing our app

For now if we change something in our code our debugger wnt restart.it will execute the previous code. You have to stop and it and run it again, if you want it to run the code that you have changed. So it would be nice if the debugger would also restart if we change our code. But with nodemon we have a package that allows us to restart. To do it-

Debug -> add configuration -> select nodejs

It will open launch.json file in .vscode folder. it allows us to configure debugging for this project and how it behaves. You can click on add configuration to see some demo settings that you can add but you can aslo add settings by typing.. We will add these seeting-

“restart” : true

You just have to make sure that you also add some other fields. For example you have to make sure that nodemon is used and for that, you set the runtime executable not to node, which will be default, but to nodemon. Now it will restart the debugger when a change is detected. You can also define here that it should start with app.js file. so that even if you have routes.js file selected while starting debugging, it will still go for app.js which is more convenient than always selecting the file you want to debug first before starting the degbugger and you always have to pick the app.js because that file starts our server. So yo cant say I am going to look into routes.js file so I will start with that.it always have to be routes.js file because you always have to start the entire server.

Lets also change where things are logged. Now things will be logged in integrated terminal. Now when you start debugger. Internal terminal of visual studio code will be used, there you can see console.log and all that stuff.

Launch.json before-

{

// Use IntelliSense to learn about possible attributes.

// Hover to view descriptions of existing attributes.

// For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387

"version": "0.2.0",

"configurations": [

{

"type": "node",

"request": "launch",

"name": "Launch Program",

"program": "${workspaceFolder}\\app.js"

}

]

}

After changes-

Launch.json-

{

// Use IntelliSense to learn about possible attributes.

// Hover to view descriptions of existing attributes.

// For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387

"version": "0.2.0",

"configurations": [

{

"type": "node",

"request": "launch",

"name": "Launch Program",

"program": "${workspaceFolder}\\app.js",

"restart": true,

"runtimeExecutable": "nodemon",

"console": "integratedTerminal"

}

]

}

Now if you run debugger it will fail. The reason for this is it will not use local nodemon but it will will look for it globally.

**npm install –g nodemon**

now our debugger works here. Now we can see all logs on visual studio code terminal(before they were in debug console). We can still ue debug console to evaluate js expressions. But in terminal you get the normal output and you have to use the terminal because if you now change something, it starts the debugger and node(in ide console you can see your code is re executed) and these are 2 separate processes. And if you quite the debugger, nodemon has to quite separately or has to exit separately and you do this by hitting ctrl+c, in console of ide and this could’nt be done in debug console which is why you have to funnel this to the terminal of ide. So that is something you have to keep in mind. You have to stop that process separately which you can do from the terminal, which is why if you are using nodemon process, you should use the integrated terminal and you can read more about that in article.

52)Debugging nodejs in visual studio code

Want to dive super-deep into the latest debugging capabilities Visual Studio Code gives you (for Node.js apps)?

This article will be very helpful:

<https://code.visualstudio.com/docs/nodejs/nodejs-debugging>

53)Changing variables in debug console

Lets s ay you want to change value of a variable when you are debugging your code. You can do it by changing the value of variable in variable section of debugger(double click the value you want to edit). It is going to affect your run time code.

55)Useful resources and links

* More on debugging Node.js: <https://nodejs.org/en/docs/guides/debugging-getting-started/>
* Debugging Node in Visual Studio Code: <https://code.visualstudio.com/docs/nodejs/nodejs-debugging>