8)Section Intro

We will start by learning moules that comes inbuilt in node. Thse are objects and functions that lets you do cool stuff with javascript, that you were not able to do before like read and write from file system.

9)Using require

Lets see module inside node. Now modules are units of functionality. So image I created few fucntimos that do math things. I could bundle those up as a module and call it sumit-math and other people can take advantage of it. Here in video we will not look how to make our own module, we will see how we can use modules. That is going to get done by using function require. Require is going to let us do 3 things-

1. It is going to load odules that come bundled with nodejs, thee are things like http server which lets us make web server and FS module which lets us access file system for our machine.
2. It lets us load 3rd party libraries, things like express which ae going to let us write less code.
3. We will use require to require our own files. It will let us break our application into multiple smaller files.

Lets see first case. You can get list of all built in modules in nodejs by going to –

<https://nodejs.org/api/index.html>

here we will sue file system to create files and od module to get user name of logged in user. If you click on any module, you will see long list of functions and properties that you have available . the one we are going to be using in this video is fs.appendFile. click on function to see how we can use it. Before calling this function we need to require it. The whole point of require is to let us load another modules. Here we will cfaete variable using const, since we re not going to manipulating the code, the module sends back, there is no need to use VAR keyword. We will use const keyword.

const fs = require('fs');

here we also use os module to get username of logged in user-

const fs = require('fs');

const os = require('os');

let user = os.userInfo();

fs.appendFile("greetings.txt", `Hello ${user.username} !` , error => {

if (error) {

console.log("Unable to red file");

}

});

10)Requiring your own files

lets create new file, notes.js-

console.log('Starting notes.js');

in app.js, import this file-

const fs = require('fs');

const os = require('os');

let user = os.userInfo();

console.log('starting app.js');

fs.appendFile("greetings.txt", `Hello ${user.username} !` , error => {

if (error) {

console.log("Unable to red file");

}

});

const notes = require('./notes.js');

note here how wehave imported notes.file. as we are importing our file we have to give relative path of this file. now lets execute app.js.

output-

**starting app.js**

**Starting notes.js**

So we see that notes.js file is also executed because we require tis file in app.js, and we executed app.js.

Now we want to export something from notes.js and use it in app.js.

Inside of all notes files, we have access to a variable called module. Lets print this object on console-

console.log(module);

output-

**Module {**

**id: 'D:\\Andrew Mead\\nodejs\\notes-node\\notes.js',**

**exports: {},**

**parent:**

**Module {**

**id: '.',**

**exports: {},**

**parent: null,**

**filename: 'D:\\Andrew Mead\\nodejs\\notes-node\\app.js',**

**loaded: false,**

**children: [ [Circular] ],**

**paths:**

**[ 'D:\\Andrew Mead\\nodejs\\notes-node\\node\_modules',**

**'D:\\Andrew Mead\\nodejs\\node\_modules',**

**'D:\\Andrew Mead\\node\_modules',**

**'D:\\node\_modules' ] },**

**filename: 'D:\\Andrew Mead\\nodejs\\notes-node\\notes.js',**

**loaded: false,**

**children: [],**

**paths:**

**[ 'D:\\Andrew Mead\\nodejs\\notes-node\\node\_modules',**

**'D:\\Andrew Mead\\nodejs\\node\_modules',**

**'D:\\Andrew Mead\\node\_modules',**

**'D:\\node\_modules' ] }**

It is pretty big object. only useful property here is exports. Exports is object on module property. Everything on this object gets exported. This objects gets set as variable in other file , where we require it-

const notes = require('./notes.js');

that means we can set properties on it and those will get on notes and we can use them inside app.js. code-

notes.js-

console.log('Starting notes.js');

module.exports.age = 25;

app.js-

const fs = require('fs');

const os = require('os');

let user = os.userInfo();

const notes = require('./notes.js');

console.log(notes.age);

console.log('starting app.js');

fs.appendFile("greetings.txt", `Hello ${user.username} , you are ${notes.age}!` , error => {

if (error) {

console.log("Unable to red file");

}

});

Output-

**Starting notes.js**

**25**

**starting app.js**

similarly we can also export functions.

Notes.js-

module.exports.add = (a,b) => {

console.log('function called');

return a+b;

}

App.js-

const notes = require("./notes.js");

let result = notes.add(8,9);

console.log(result);

now run app.js

output-

**function called**

**17**

11)Using 3rd Party modules

Lets install some 3rd patry packages using npm. Navigate inside project folder. run this command-

**npm init**

this will create package.json file in your project. This command will ask you some questions. answer in () is default one, press enter to select that. Questions-

**package name: (notes-node)**

**version: (1.0.0)**

**description:**

**entry point: (app.js)**

**test command:**

**git repository:**

**keywords:**

**author: sumeet sood**

**license: (ISC)**

package.json is simple description of your project. right now it has just details that we have filled while creating it. As we are not going to publishing our app to npm, so lot of this information here is not relevant to us. What is important is that package.json is where we define third party packages that we want to install in our app.

Here we will install lodash. Lodsah comes with ton of utility methods and functions that makes developing inside node or javacsript, a heck of lot easier. Go to npm site. Then search lodash in package searh text box. When you click, it would take you to package page and package page is going to show you a lot of statictics about module and documentation. Here see how many downloads package has and when was it last updated. If it was updated recently then it means it is compatible with latest version of node. In downloads you can see that it is one of most popular package, with over million downloads a day. Now run this command-

**npm install lodash --save**

--save is going to update package.json with this package name. a new property called dependency will be created in file and as avalue it will have name of package and their version.now our package will be downloaded in node\_modules folder. if it is not there then it will be created. Now lets see how we can use this package inside app,js-

const \_ = require('lodash');

lodash is sname that appreasin package.json and it is name that you used in install command. \_ is common name of lodash library.

First node will look for core module named lodash. It won’t find any, then it is going to look into node\_mdolues folder. So now we have lodash imported, lets use it. Now lets go through its documentation. On npmwebsite on top there is link to lodash website. On this website go to this link-

<https://lodash.com/docs/4.17.10>

use ctrl+f to serach.

Lets explore some methods of this library.

const notes = require('./notes.js');

console.log(\_.isString(true));

console.log(\_.isString('sumeet'));

this methods removes all duplicate values from array-

let filteredArray = \_.uniq(['Andrew', 1,'Andrew',1,2,3,4]);

console.log(filteredArray );

lodash library is limitless. Use this when possible. All functions here are tested.

Then we talked about npm install command. How wit installs our packages, we should never send node\_modules fiolder while sending our project.

12)Restarting app with Nodemon

Here we installed tnodemon package globally.

npm install -g nodemon

it will automatically re run our files when we change something. To execute our project runthis-

**nodemon app.js**

you will see outputs as well nodemon logs like version, files it is watching and command it actually ran.

13)getting Input from user

Code-

const fs = require('fs');

const \_ = require('lodash');

// const notes = require('./notes.js');

let command = process.argv[2];

console.log(process.argv);

console.log('Command', command);

if (command === 'add') {

console.log('Add');

} else if (command === 'list') {

console.log('Listing');

} else if (command === 'read') {

console.log('Reading notes');

} else if (command === 'remove') {

console.log('removing note');

} else {

console.log('Provide some command');

}

Output-

**[ 'C:\\Program Files\\nodejs\\node.exe',**

**'D:\\Andrew Mead\\nodejs\\notes-node\\app.js',**

**'add' ]**

**Command add**

**Add**

Here we are getting argument from usr and printing it on console.

So we can run commands like this – **node app.js add**

So we are taking action depending upon third argument. now what I want to do, I want to pass . so this is step one. Next step is we are going to figure out how to get more specific information, for ex which node do you want to remove, which note you want to add, what should be text of newly created note. Like this-

node app.js remove --title=secrets

in this command we are passing title argument into our application.there are couple of different ways you can format this.

node app.js remove --title=”secrets”

or

node app.js remove –title ”secrets”

now depending upon how we run this command, contents of argv will be different, so we are not going to extract argument value from argv array.

So parsing certain types of command line arguments like key value pairs becomes a lot more complex which is why in next video we are going to be using yarns to do just that.

14)simplified input with yargs

Install yargs package-

**npm install yargs –save** or we can install specific version like this-

**npm install** [**yargs@4.7.1**](mailto:yargs@4.7.1) **--save**

yargs is very complex library. It has ton of features for validatimg all sorts of input and it has different ways you can format that input. We will start with very basic example and we will be introducing more examples.

Now lets fetch the arguments as yargs parses them.Yargs is taking same process.argv array but its going behind the scene and parsing it giving us something that’s more useful than what node gives us. So we do this-

const yargs = require('yargs');

// const notes = require('./notes.js');

const argv = yargs.argv;

argv is where yargs stores its version of arguments that you app ran with. Lets run –

const argv = yargs.argv;

let command = process.argv[2];

console.log('Command', command);

console.log('Process', process.argv);

console.log('Yargs', argv);

command –

node app.js add ecrypted

output-

**Command add**

**Process [ 'C:\\Program Files\\nodejs\\node.exe',**

**'D:\\Andrew Mead\\nodejs\\notes-node\\app.js',**

**'add',**

**'encrypted' ]**

**Yargs { \_: [ 'add', 'encrypted' ],**

**help: false,**

**version: false,**

**'$0': 'app.js' }**

Yargs gives us object which looks very different. it gives us \_ property , this is where our commands are stored.

This does not seems very useful. But yargs becomes useful when we start passing property value pair.

**node app.js add encrypted --title=secrets**

**Command add**

**Process [ 'C:\\Program Files\\nodejs\\node.exe',**

**'D:\\Andrew Mead\\nodejs\\notes-node\\app.js',**

**'add',**

**'encrypted',**

**'--title=secrets' ]**

**Yargs { \_: [ 'add', 'encrypted' ],**

**help: false,**

**version: false,**

**title: 'secrets',**

**'$0': 'app.js' }**

So here we got property title sand its value as secrets. While where in process.argv we giot whole string. This parsing of yarg will work, we can pass property value pairs by all 3 methods and yarg will give this same output.

In notes.js we are exporting a function. here we will use different syntax for exporting-

let addNote = () => {

}

module.exports = {

addNote: addNote

};

Here object property and value, both are same. In es6 there is shortcut for this-

let addNote = () => {

}

module.exports = {

addNote

};

We will be using Es6 syntax through out our course.

15)Working with Json

Here we will store notes in a file and fetch from there. For this we need to introducing something called json. It is way to represent javascript arrays and objects using strings. Why would we do that? Because strings are just texts and that’s pretty much supported anyway. I can save json to a text file and then I can read it later, parse it back into javascript array or object and do something with it. Code-

let obj = {

name: 'Andrew'

};

let stringObj = JSON.stringify(obj);

console.log(typeof stringObj);

console.log(stringObj);

let obj2 = JSON.parse(stringObj);

console.log(obj2);

console.log(typeof obj2);

output –

**string**

**{"name":"Andrew"}**

**{ name: 'Andrew' }**

**Object**

When we convert object into json its type is string. In json string are in “ instead of ‘.

Lets write a code where we are saving note object in file the we are reading tht object from file and converting it back into js object. code-

const fs = require('fs');

let orignalNote = {

title: 'some title',

body: 'some body'

};

let orignalNoteString = JSON.stringify(orignalNote);

fs.writeFileSync('notes.json', orignalNoteString);

let noteString = fs.readFileSync('notes.json');

let note = JSON.parse(noteString);

console.log(note);

putput-

{ title: 'some title', body: 'some body' }

16)Adding and saving notes

Here we implemented the addNote function. note.js-

const fs = require('fs');

let addNote = (title,body) => {

let notes = [];

let note = {

title,

body

};

try {

let noteString = fs.readFileSync('notes-data.json');

notes = JSON.parse(noteString);

} catch(e) {

}

let duplicateNotes = notes.filter(note => note.title.toLowerCase()

=== title.toLowerCase());

if (duplicateNotes.length === 0) {

notes.push(note);

fs.writeFileSync('notes-data.json', JSON.stringify(notes));

} else {

console.log('Duplicate Note already exists');

}

}

let getAll = () => {

console.log('Geting all');

}

let getNote = (title) => {

console.log('Getting note', title);

};

let removeNote = (title) => {

console.log('removing note', title);

};

module.exports = {

addNote,

getAll,

getNote,

removeNote

};

We have placed the code where we read the file in try block. If we do not do this then we will get error when file does not exist., so our app will break After placing that code in try, now if we get error, then control will pass to catch black, note that statement after catch block will be executed.

One more thing is that when we write to file, we remove existing contents. So we have wrote the logic of getting existing notes and then pushing new note , then saving it.

20)Debugging nodejs applications

Here we saw that how we can dubugg node applications using cmd command. For this node version 8 or higher is need. Althrough it is dnt find it useful, its combination with nodemon is awesome. See last of the video. lets debug our debugging.js file. run this-

**node inspect debugging.js**

output-

**b< Debugger listening on ws://127.0.0.1:9229/10fb73fa-8b07-4b68-b798-89a4fde25a91**

**< For help, see: https://nodejs.org/en/docs/inspector**

**Break on start in debugging.js:1**

**> 1 (function (exports, require, module, \_\_filename, \_\_dirname) { let person = {**

**2 name: 'sumeet'**

**3 };**

**debug>**

first 3 lines just tell us that debugger was successfully setup. Now our application is not started.

There is command that you can run in debug mode- **list(n)**. here n is number of lines to be shown above and below where we are paused. Now our code is sitting inside a wrapper function, this wrapper function is actually created by nodejs and every single line of code that you write in nodejs is wrapper inside this function. it gves us access to require, module and other things.

If you want to next line presss **n** and **enter**.

If at some point in time you want to continue execution press **c** and **enter**.

If you want to see value of expressions inside debug mode, press **repl and enter** , now you can see value of any expression, change value of expressions. it is just like console in chrome dev tools. To get out from repl mode press ctrl+c. now you will in in debug mode.

Now lets say we have very big app and we want to stop execution at middle. Now we cannot go line by line. To do that write debugger statement before that line in code. In cmd press c, your execution will stop at debugger.

We can also use this debugger with nodemon. Just replace node with nodemon.

21)Debugging via chrome dev tools

Chrome version 8 or higher is required.

Things like debugger will also work I chrome dv tools. Run this command-

**node --inspect-brk debugging.js debugging.js**

now go to chrome and type this in url bar-

**chrome://inspect**

there click on **open dedicated devtools for node.**

note that initially only one file will apper in sources tab, other files will apper as they are used.

again we can use this with nodemon . just replace node with nodemon.

Note that best tool to debug node is visual studio code. Lets we want to pass comqamnd and command line parameters while debugging with visual studio code. To do that add args property in configurations array. Like this-

"configurations": [

{

"type": "node",

"request": "launch",

"name": "Launch Program",

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

"args": [

"add",

"--title=degug",

"--body=lastbody"

]

}

]

23)Requiring Arguments and Advacnced yargs

Here we saw how we can define help menu of our app.

24)Arrow function

We know there are 2 differnces. For details see js course of Andrew. When we print arguments inside arrow functions in nodejs we will see a very big object. this arguments is the argument of function in which all our node code is wrapped.