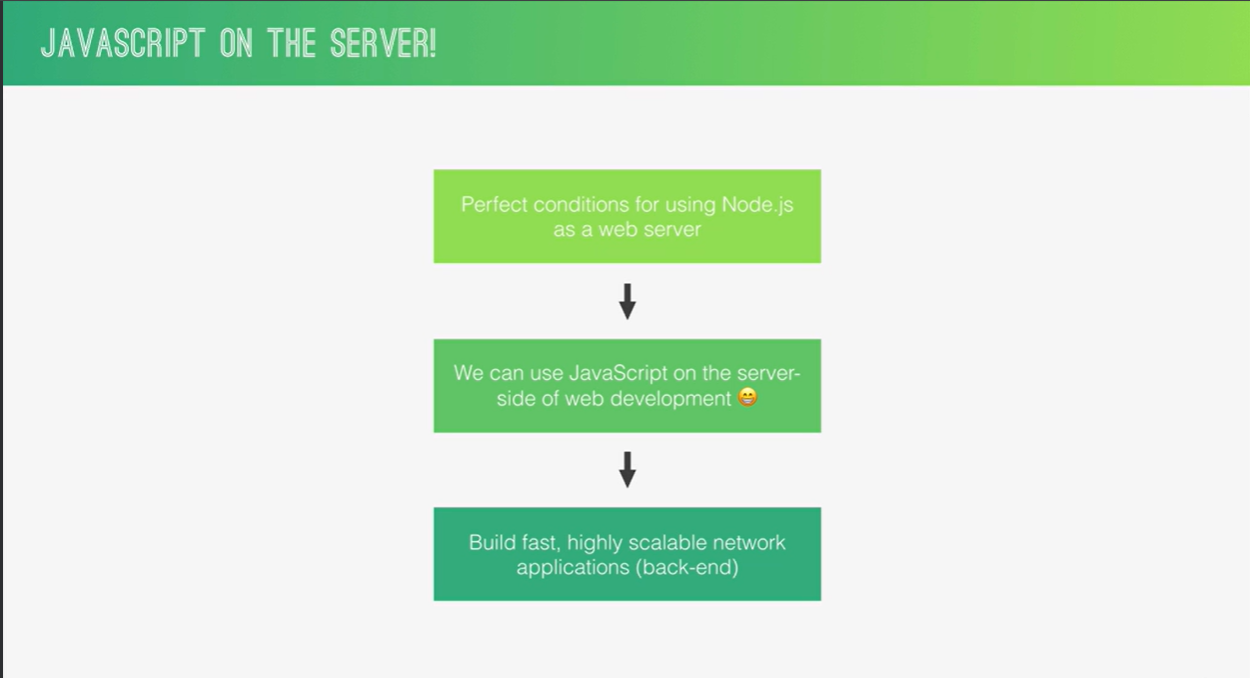
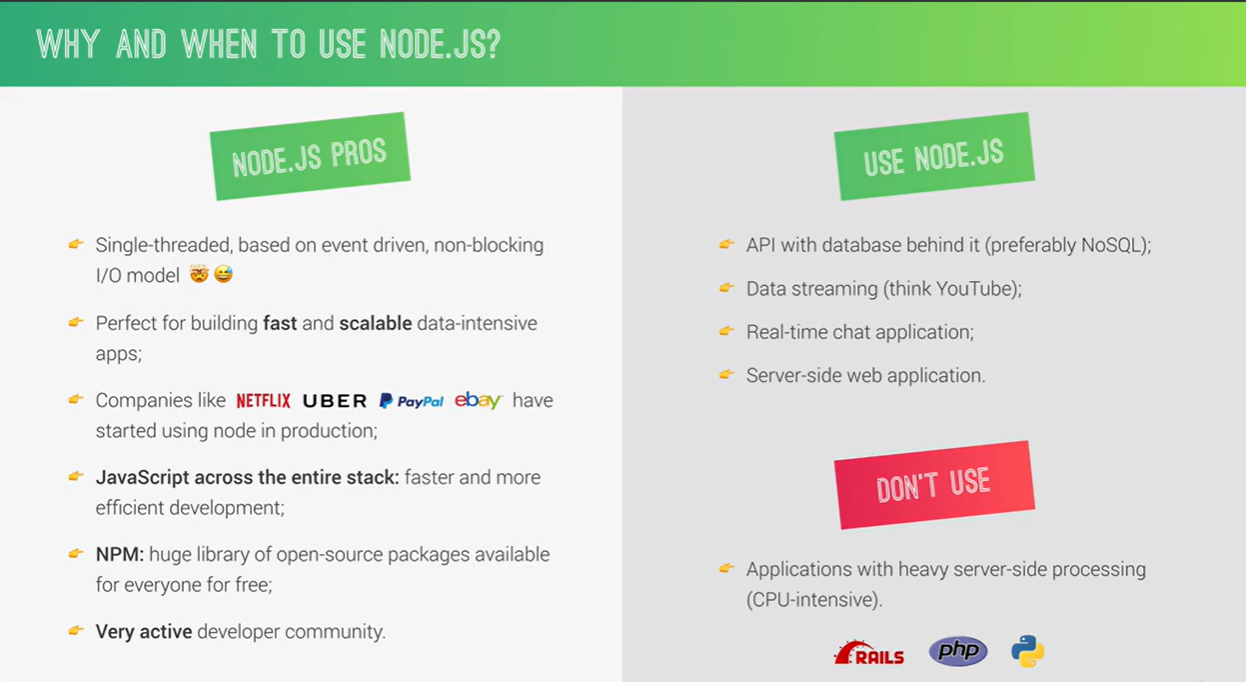
The Basic Definition of Node js is that it is an upper version of JavaScript which is used to run and execute the JavaScript code outside of the browser which is not possible without Node js. It allows us multiple more features with js like it enables us to use it at backend also and it allows us to create the server.



**Why and When to user Node JS ?**



Important **We can check node js version by : node -v**

Once we make our folder structure and in that folder gave the command : 'node' then it will initialize it as node repo and we can write our code in the terminal and execute the code like we can do in any other terminal.

To exit this repo we can press Ctrl+d or we can give command '.exit' they both will allow us to exit the node repo. Inside the node repo we can press tab (sometimes twice) to see all possible data types or operation. Like if we enter the tab just on the clear screen that it will show all data types available there and if we press tab after writing 'String.' then it will show all possible functions or operations over String .

Underscore '\_' will give us the last value : if we do 10+3 then it result 13 will be stored and show on the terminal and after this if we will do \_+3 then it will give us 16 because for '\_' it will take the last value that is 13.

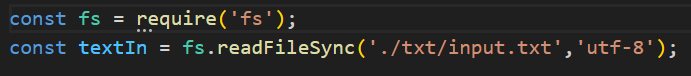
To run any js file in our terminal we need to give the command 'node fileName.js' and now instead of making one HTML file and then integrating the script file in html and then see the result in browser, we can just simply gave this command and execute the code there only.

Node Js provides us a lot of feature that were not possible with JS, and one of the main feature is to read the file from our disk and perform operations on that.

**Note :** To take a file from disk we need to import a module 'file system' syntax : const fs=required('fs').

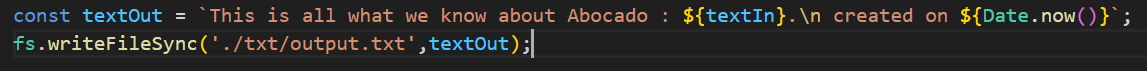
To read the file from disc we need to use readFileSync function of file system(fs) and this function use to take two value, first : path of the file which we want to provide in input and second character encoding. And it use to return the text from the file.

For ex : const text = fs.readFileSync('/txt/input.txt','utf-8');



To write on the file on our disk, we need to use writeFileSync function of file system(fs) and this function use to take two value first path of the file and second text that we want to write on the file. It don’t return anything.

For ex : const text = 'This is the text that I am writing to output.txt file for its testing whether it will be written or not.' fs.writeFileSync('./txt/output.txt',text);



**Note** : Sync in the file name here is synchronous, we have asynchronous function also, we will see it in upcoming sections.

**Synchronous vs Asynchronous Code ( Blocking vs Non-Blocking) :**

**Synchronous(Blocking Code) :** So Synchronous Code is also known as blocking code because it executes line by line, what we mean by it is that once the code on line no 2 will be finished only after that line no 3 will be executed and same will go ahead further. So each line have to wait for all its upper line to get executed and which result into the slowness in our application.

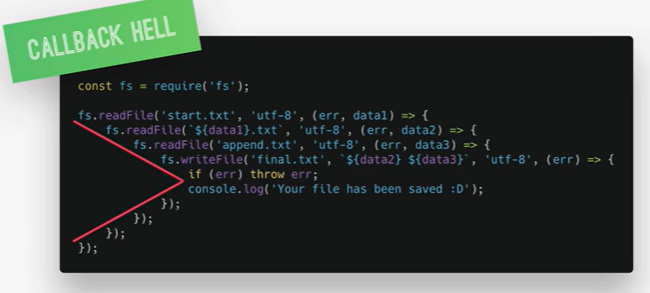
**Asynchronous(Non-Blocking Code) :** Asynchronous code is like where when a block of code will be executed in async and the code after that can still continue executing, It don't have to wait for its upper lines and it reduce the slowness in application.

We will see the Async function to read and write on file in coming section.

Important **Note :** Node JS is a single thread language so how asynchronous code works in it. If a line is getting executing that may require more time we will use callback function there, what happens there is the task will be getting executed in the background and other tasks or code can still continue their executing so that application won't be sucked due to of that and once that bigger task will be completed the call back function will return that task to the single thread i.e. main thread. That's why we will see that there are so many callback functions will be used in Node JS. And if we are using callback function, it doesn't always mean Asynchronous function, because callback function are very common in js also.

What is callback Hell ?

: When we are reading and writing in a multiple layer in async function then it's said to be callback hell.



Here you can see its just 4 layers and how much complicated it seems, think if there are 20 layers then how much complicated it will become to write and understand, so this is known as callback hell.

Important To Escape from this callback hell we can use Promise or Async/Await from modern JavaScript.

**Asynchronous function to read :** To read the file from disk readFile is the async function, it use to take three value, first path of the file, second character encoding and third callback function which will take two param first err and second data.

A screen shot of a computer code

Description automatically generated

Here first it will print reading the file and then it will print the data, this is the async function.

A black background with white text

Description automatically generated

**Asynchronous function to write :** To write on the file writeFile is the async function, it use to take 4 values, first path, second text to write, third character encoding and fourth callback function that will take one param only err as it don't have any value.

A screen shot of a computer code

Description automatically generated

Important **Creating a server :**

To create a server in Node JS we need to import http module : const http = require('http');

After importing the http module we can create a server on it with function createServer(), this function will take a callback function which takes two inputs request and response, for as of now for basic example we are just adding a simple response and sending it back as response using .end function.

A black screen with white text

Description automatically generated

And now we need to enable to port and host for this server so we will use listen function on this server which takes two input port and host :

A black background with white text

Description automatically generated

This callback function is optional, it is like If we want to perform some operation as the start point of this server. Now this function is running on localhost:8080 and we can check it by hitting this port.

A screenshot of a computer

Description automatically generated

We can start this server with 'node fileName.js'.

And to close this server we can press Ctrl+C.

**Routing :** Actually routing becomes much complicated in real world big application, so we use a tool for that 'Express' but as of now we will learn how can we do the routing through Node only and later we will learn through Express.

For Routing to analyze the URL we need to import the module url : const url = require('url');

For this Basic routing we just need to take the URL from request and use the if else block to divert the response. Exp:

A computer screen shot of text

Description automatically generated

Here we are just checking the path and diverting the response accordingly, if we want to add status also then we can use res.writeHead, It can receive two params, first status and second an object and inside that if we want to send any other header we can put them there. Here as we gave Content-type as html/text then we can send the response also as html.

ImportantHighlight **Very Important Note :** If we are giving the path for any file and we are using ./text like this so here '.' means from where we are running the application i.e. from where we are giving this command 'node fileName.js' so may be if we are running the server from desktop then this path will become incorrect. So to overcome this issue we have one more option, so we will use '\_\_dirname'(2 time underscore + dirname) (and we need to take this as a variable inside the back tick) instead of '.' this \_\_dirname will always take path from where file is placed i.e fileName.js



**Building First Very Basic API :**

So we are reading the data from a file and sending it back as response on hit of an API, Ideally we should do it like this using an async function :

A screen shot of a computer code

Description automatically generated

But here as we are reading this data from the file so this is not an ideal way to this because here we are reading the same data from the file again and again so there is no sense of doing this so here we will you the Sync function to read the data once and then we will just simply send that data as response from our API.

A computer screen shot of a program code

Description automatically generated

**HTML Templates for NODE JS :**

To make our dynamic templates with node js, we need to place some placeholders at the part in html which we want to replace with our data dynamically, so that we can replace those placeholders according to the requirement and can get the expected results.

A screen shot of a computer code

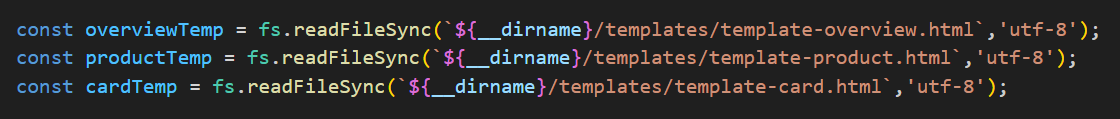
Description automatically generated

As seen in the above example we can put any name there for placeholder, what we need to focus and understand is that it should be unique, if we want same value at multiple place then we can place the same placeholders else it should be unique only.

**HTML Templates filling with Node JS :**

Now to fill these html templates that is created above with the real data set, that is to replace those placeholders which we defined above we need to first read this file(template file) with file system and then we will take the text out from it and will replace these placeholders values in our text with the original value that need to be filled.

So first we will read this file :



Then we will make a function which will take this text and our product as input, and will replace the product's data into this text and will return the same. So we will call that function by passing this text and original value.



So here we are passing productTemp(text of product template) and prod\_data[a], this is an array and we are sending the element at a-th index, so that it can fill the a-th element's values in the template's text.

This function seems like this :

A screen shot of a computer program

Description automatically generated

So inside replace function we are using like this '/{%PROD\_NAME%}/g' so that it can replace all the occurrences of this particular text with the provide one i.e. data.productName .

Then we can set this returned output in the response to achieve the expected results. We also need to set the content-type in the header to make sure that we are sending the HTML.

A black screen with text

Description automatically generated

Note : In case if we are using a template inside which there can be zero, one or more products can be possible there based on the numbers we have in our backend or db, then we will put a placeholder in that home page template like this :

A black screen with white text

Description automatically generated

So after placing this placeholder here, we will take that array of products and will replace one by one all values with the array iteration, and then we can add all those value as a string or we can leave them as an array also, and then we will replace this placeholder in home page with that updated string or array, and will send this home-template in response to achieve the correct output.

A screen shot of a computer code

Description automatically generated

So here we have an array prod\_data, we used the map method to iterate over the array and replace the card template with all the values one by one, so in cardArr we will have an array with all the updated templates of cards, so once we have that updated array, we will replace that in our home page, to show all the results dynamically and then we can send this home template in the response.

**Parsing Values from URLs :** To parse the values from the URL i.e. to find the query params from the URL, we need to use the module 'url' which we have already imported in some upper section. So once on this URL module we will parse the request url, it will give us all the value that are possible inside that URL. So for query params it will give us an object 'query' which will have all the fields which we have passed. And it will give a value pathname which will have path or the url(except query params).

A black background with blue and green text

Description automatically generated

This is an ES6 feature which will automatically declare this two variables and will set the value from this object to the respected variables, like it will assign query the value of object.query and same for pathname.

A black background with blue and white text

Description automatically generated

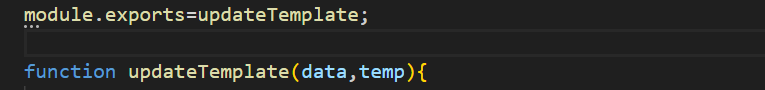
Like here we matched the pathname for routine and we took query.id for getting the id from the url. The URL passed here was like this '127.0.0.1:8080/product?id=1', so pathname gave us '/product' and query gave us id=1;

**Creating our own custom Module :**

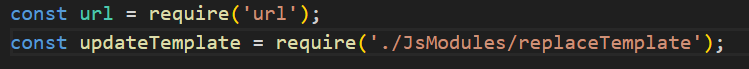
So, let us first understand why do we need to create a custom module, as we have seen above we are using a function updateTemplate to replace the placeholders with the real values to achieve dynamic pages. So as of now we are doing it in a single file, so let's say we need to update the templates in some other files also so in that case we need to again create the same function and use there, so instead of doing all this task what we can do is, we can create a module, and we can create this function there and can export from there, so that we can just import that module in our file and can direct use this function.

Now, let us understand how to create a custom module and how to export import its functions. So in Node JS, all java script files are automatically treated as module, they all have access to a variable 'module'.

So we can just simple create a JS file and create our function in that class and can export that with help of module variable like : module.exports = updateTemplate;



So like this we can export our function from our module, now let's see how can we import this custom module.



So I gave this name as updateTemplate only, so that we don't have any need to make change in my codebase, and our code will still work fine. Otherwise we can give any name here and same we can use. Here we always have to give the path from '.' only, don't user '\_\_dir' here.

**NPM :- Node Package Manager is used to install packages in our application.**

**First we need to give : 'npm init' to initialize the project as npm repo.**

We can install two type of packages through NPM (1. Simple/Regular Dependency and 2. Development Dependencies)

To install any package we just need to give : npm install packageName .

Simple/Regular Dependency : - The dependencies which we used to enhanced are coding part, and for these dependencies our code will be dependent on them. Ex : slugify (npm install slugify)

Development Dependency : - The dependencies which are basically used in development and not in production i.e. our code will not be dependent on this dependencies. Ex nodemon (npm install nodemon --save-dev) we have used -dev here to tell that this is an dev dependency. This nodemon help to auto restart the server whenever we made any change in the working directory, so we don't need to stop and start the server, it will do it for us automatically.

To install any package globally, we can give npm install nodemon --global. It will install the nodemon globally and we don’t need to install it again in other projects. We use to install these type of package globally, which we need in all projects, so that we don't need to install this dependencies in every project separately.

No to let our app auto restart after any change in our code, we need to start the app with nodemon, means now we need to give command **nodemon fileName.js** instead of node fileName.js. So now our server will be auto restart as soon as it will pick any change in our codebase.

When we give command it npm init i.e. when we initialize any project as npm repo, then it creates a file package.json which contains all info about our project. So there is one object 'script', so if we will set 'start' = 'nodemon fileName.js' there and then we can simply give the command npm run start, then it will start our application with nodemon.

A black screen with white text

Description automatically generated