Module Pattern In Node Js

22 June 2023 01:10

Module Pattern in NodeJS –

Module is a mechanism for splitting JS programs into separate manageable chunk called module that can be imported whenever needed.

Inside Nodejs we have two mechanism to prepare a module

(i) CJS (traditional mechanism) – Common JS Module
(ii) ESM – ES6 Module

CJS – Demo

searching.js

```
const linearSearch = function search(arr, x) {
    for (let i = 0; i < arr.length; i++) {
        if (arr[i] == x) return i;
    }
    return undefined;
}

const binarySearch = function search(arr, x) {
    let lo = 0, hi = arr.length - 1;
    while (lo <= hi) {
        let mid = lo + Math.floor((hi - lo) / 2);
        if (arr[mid] == x) return mid;
        else if (arr[mid] < x) {
            lo = mid + 1;
        } else {
            hi = mid - 1;
        }
    }
    return undefined;
}

module.exports = {
    linearSearch,
    binarySearch
}
```

To use the module in a different file, we can use the global module to export a bunch of functions

global

This object is now going to be exposed to the outer world

In any other file we can start using it

it's a shorthand used since the key & the value name is same

this can also be written as
ls : linearSearch
bs : binarySearch

index.js

```
const { linearSearch: ls, binarySearch: bs } = require('./searching');
console.log(ls([3, 2, 1, 5, 4, 23, 6], 23))
```

destructured object

alias name

It's another global, it's kind of like an import syntax, whatever are the modules that we are exporting from the file (for now it's searching.js) in the particular path we will get them all and we are storing it inside a variable

Benefit of destructuring is that whatever function or property we only those property is getting

whatever func... need only those property is ge... used.

Note: Destructuring means unpacking

2 other way to export module

```
module.exports.insertionSort = function insertionsort(arr) {
    for (let i = 1; i < arr.length; i++) {
        let currElement = arr[i];
        let j = i - 1;
        while (j >= 0 && arr[j] > currElement) {
            arr[j + 1] = arr[j]; // shifting the element to the right index
            j--;
        }
        arr[j + 1] = currElement;
    }
}
```

```
module.exports.insertionSort = insertionsort
module.exports.bubbleSort = bubblesort
module.exports.selectionSort = selectionsort
```

Now converting them to ES6 Modules

↳ There are two ways → adding extension .mjs
→ updating type of the module

By default nodejs treat the following as CommonJS

→ .cjs → .js

So, for ES6 modules we need to configure package.json file and need to put the key as the type we want. If the key is "commonjs" then we enabled commonjs module although commonjs is the default.

Now if we put type = "module", things start to change

Through .mjs

exporting module

```
export function insertionsort(arr) {
    for (let i = 1; i < arr.length; i++) {
        let currElement = arr[i];
        let j = i - 1;
        while (j >= 0 && arr[j] > currElement) {
            arr[j + 1] = arr[j]; // shifting the element to the right index
            j--;
        }
        arr[j + 1] = currElement;
    }
}
```

importing the module

defining path mentioning its getting imported from cjs file

```
import searchingAlgo from './searching.js';
// import * as sorting from './sorting.mjs'; //it brings all the named export and club them in the variable sorting which is technically going to be a object and then we can access as sorting.bubblesort(arr);
import Mergesort, { bubblesort as bs } from './sorting.mjs';
```

default import

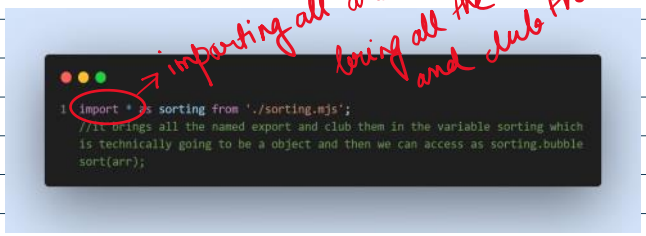
deconstruct import with alias

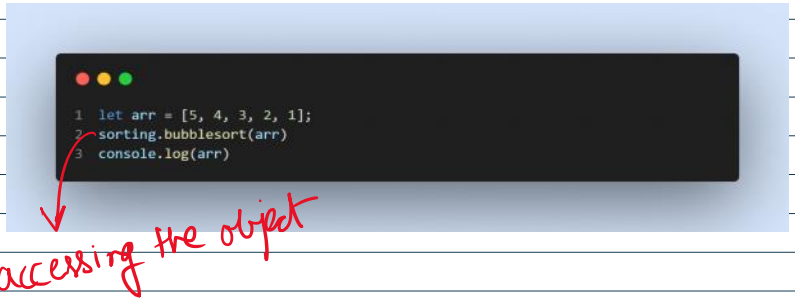
file type esm

default export

```
export default function mergesort(arr) {
    return f(arr, 0, arr.length - 1);
}
```

named export together insorting





⇒ NPM – Node Package Manager

Multiple modules clubed together to form a complete end to end functionality is called a package.

There are multiple packages we can find in NPM.

What NPM help us to do is, it help us to install the packages and even do dependency resolution

for example– Let's say there's a package (A) that is dependent on package (B) might be using some functionality of it but we don't know about it

When we install (A) it'll directly manage package (B) also. It will do topological sorting of all the dependencies and install everything one-by-one

→ We can very easily download module by running the module name with npm

ex – npm i dotenv

The moment we enter this the module gets downloaded, when we check our code editor we don't just see the module we wanted we see

3 other things as well – (i) Node modules – it has all the dependencies present that our install module require

(ii) Package.json – It help us to understand metadata about our project.

for ex-

(i) Who this project is getting prepared for

(i) Who this project is getting prepared for

(ii) What are the dependend packages or the dependencies

(iii) Are their any script configured for this project

(iii) Package.lock.json — It has record of all the dependency of our installed module

on — Name of the dependency
Version "
etc

We can prepare our very own package.json from very scratch what we need to do is execute a command 'npm init'
what npm init does is, it creates a package.json file for you
technically saying it helps in creating npm project.

↓

project having a package.json file is a simple npm project

Project 0 - Telegram Bot

↳ npm init > basic config

↳ in package.json file add type = "module" or "commonjs" whichever modeling pattern you want in your project. By default it's commonjs

↳ Copy telegraph npm & paste it, it'll install telegraph for me

↳ Create a file index.js

↳ How to use telegraph or any other node module?

It's the documentation we should always refer

↳ In the index.js file add

↳ In the index.js file add

```
const { Telegraf } = require('telegraf');
```

using absolute path

CommonJS

↳ Now in order to create a new bot, we need bot token's from telegram

→ All we need to do is open telegram and search for botfather

→ Botfather is a telegram bot that helps in getting telegram bot token.

What is bot token?

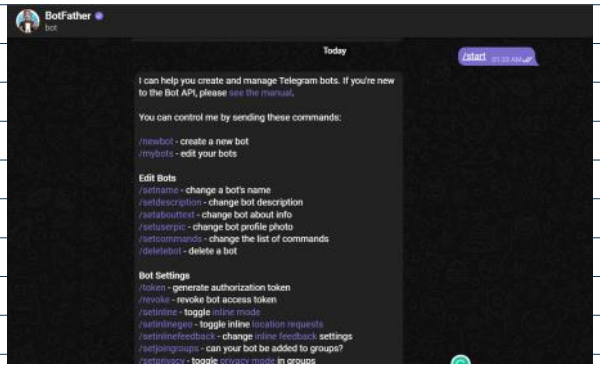
→ Token through which telegram will be able to say who owns this bot

→ A lot of time when you'll use 3rd party services they will give you similar token as API key, API token. Using these token they are actually identifying who is using the particular app

→ In BOTFATHER → /start

anything with "/" treated as command.

it replies with multiple commands



↳ to create a new bot enter /newbot

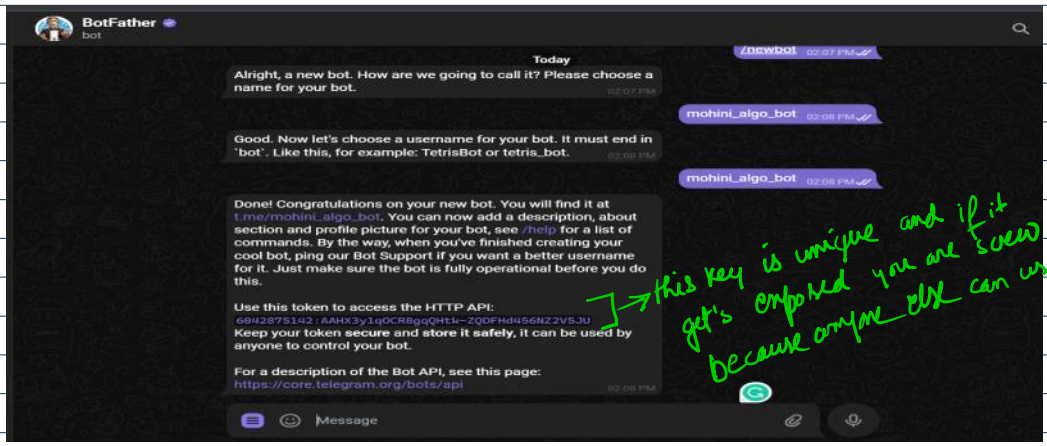
↳ It'll instantly ask to name the bot, enter the name of the bot whatever you want

username

whatever you want

↳ Done our bot is ready it gives a link through which anyone can access the bot

↳ With the done message we also get a API key, through which we can customize the bot however we want now



this key is unique and if it gets exposed you are screwed because anyone else can use it then

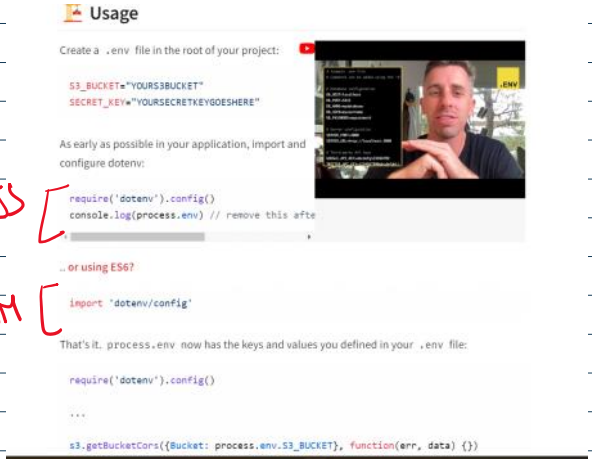
So, there are mechanism through which we can hide our secret key, we can actually store them in environment variable.
Every process we run has a key value pair in operating system and we can add more key value pairs. How to do that?

→ that's what the dotenv package do.

So now npm install dotenv

→ Once dotenv is installed create a (.env) file inside the project

→ Inside the .env file we can write secretkeys, password, emails etc with a name you want to use it as.
Now to import you can use both CJS version, ES M version depending on the project as well



To use it, we need to use "process.env"

variables

to use it, we need to use process.env

global, it helps to access all the env variables

Once our env variable that we want to use is BOT_TOKEN

∴ const bot = new Telegraf (process.env.BOT_TOKEN")

This is how we can actually hide our secrets

↳ How does .env file secure this file?

⇒ Wherever we push the code, we will not push the .env file using .gitignore file. But let's say you want to deploy it on the server. On that server we have to manually make this .env file or we need to prepare a pipeline that can manually push the .env file

So, this is how we create a bot & we are good to go

→ Now, to start the bot

```
bot.start((ctx) => ctx.reply('Welcome'));
```

↳ telegram bot object

in the telegram bot chat if we enter /start, the bot will get activated

↳ to give a particular let's say /binarysearch

bot.command ('binarysearch', (ctx) ⇒ ctx.reply (binarysearch Str

↳ event, it's for custom reply

↳ Configuring an emoji

bot.on('sticker'), (ctx) ⇒ ctx.reply ('😍'));

↳ event

↳ At the end when the bot is launched

bot.launch()

↳ To handle messages we will bind them inside try catch

Code Snippet -

Functional Chat -