# **Introduction to Node.js?**

* Node.js is an open source, cross platform runtime environment and library of JS.
* Used for server-side programming
* Primarily developed for non-blocking, event-driven servers
* It is built on Google Chrome’s V8 JS engine
  1. **What is node.js?**
* Node.js app runs in a single process, without creating a new thread for every request
* It provides set of asynchronous I/O (primitives = ancient) in its standard library to prevent JS code from blocking
* engine
  1. **Why use node.js?**
* Due to huge community
* V8 runs JS code lightning-fast
* Node.js applications are very scalable and maintainable
* We can create desktop app, software as a service, mobile application due to cross platform support
* Node.js is perfect for real-time and data-intensive application
* Huge collection of node packages (npm)
  1. **History of node.js**
* Written by Ryan Dahl in 2009,
* Developed for Linux and macOS
* Later sponsored by Joyent
  1. **Node.js vs browser**
* We mostly interact in Browser with DOM and cookies, which don’t exist in Node.js
* In Browser we don’t have nice APIs that Node.js provides through its modules
* In Browser we cannot choose the version but in Nod.js we can control the environment
* Browser uses Babel to transform JS into latest JS code but Node.js supports all modern JS code (ES6+ ,2015>)
* Node.js supports commonJS and ES6 module system
* We use import in Browser and require and import in Node.js
  1. **Running node.js code**

At the same directory open command prompt and run the file with this code

“node app.js” if the file name is app

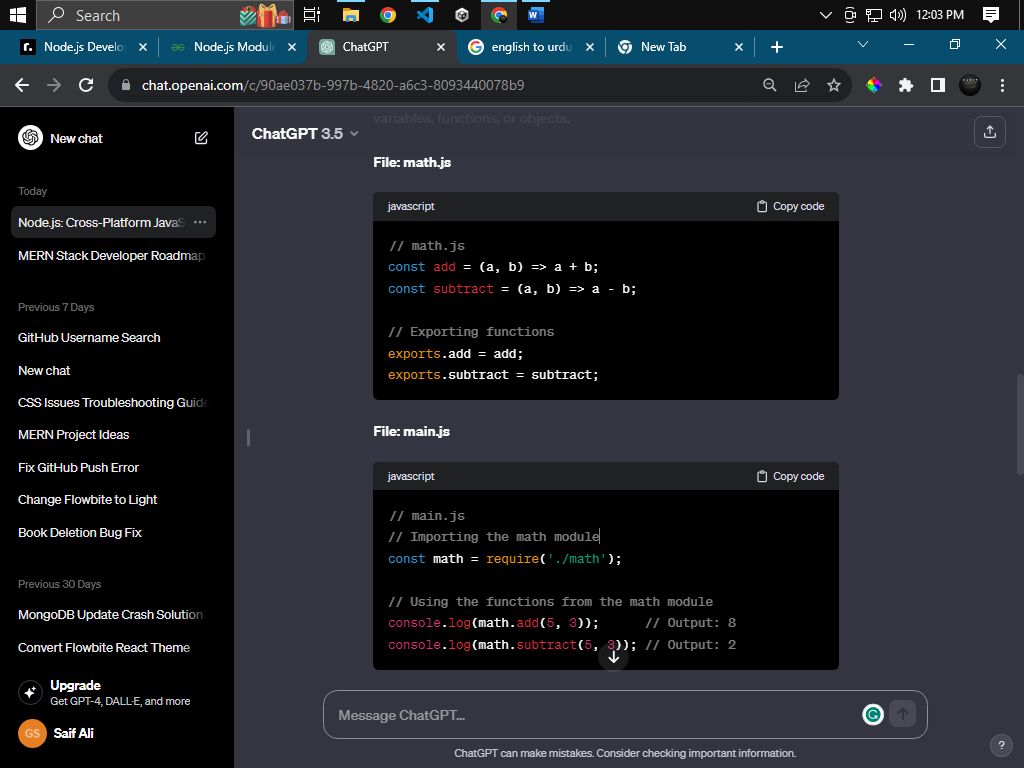
And can write some code and run at the command prompt using “-e”

“node -e “console.log(‘Code is running using -e command in command prompt’)””

# **Modules**

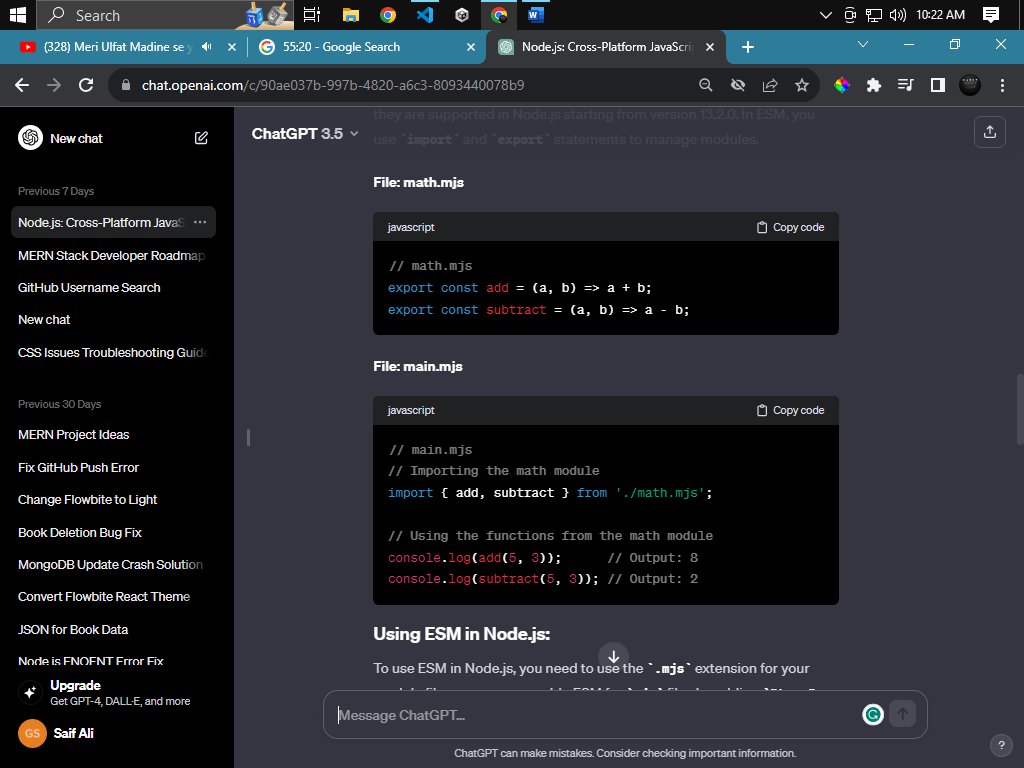
* Chunk of codes to maintain, organize and reuse as needed
* They are JS files written by other developers
* Some built-in modules that are part of platform and come with Node installation (http, fs, path, querystring, process, os and path)

1. **Common JS**

* Traditional modules those are exported by exports.module name and imported by using require()
* Not accessible to others by default

1. **ESM**

* Advanced modules, introduced in version 13
* Instead of exports.module we use export keywords
* And import in another file like react (import)
* We can assign any name if export as default
* Import and export name must be the same



1. **Creating Custom Modules**

Have done in a and b.

1. **Keywords [global]**
   * 1. Global objects

Global, console, process, buffer, setTimeout, setInterval, clearTimeout, clearIntervel

1. **Global Keywords**

Var, let, const, function, return, if, else, switch, case, default, while, for, do-while, module, exports, import, require, try, catch, finally, async, await, new, this, typeof

# **NPM**

1. **npx**

* Node Package Manager
* Online repos for publishing open-source Node.js projects
* Worlds largest software registry
* Run packages without downloading using npx.
* It is a command line utility to interact with the installed package
* Use to manage and share software packages or libraries in JS code.
* It simplifies the process of installing, updating, and managing dependencies in node.js project
* Developer uses npm to install packages (libraries or modules)
* Npm ensures that the correct version of package is installed to avoid the conflict
* “npm install” is used to install dependencies and “npm init” to create for “package.json” file
* We can set the version in “package.json” to allow the specific version
* We can create custom script in “script” to do specific tasks
  1. Global Installation
* Node and npm allows us to install any package globally or locally in the system
* We can install within the project that will work run only in the project
* And we can install the package globally within the system environment using this “-g” and can use with any project within the system

1. **Local Installation**

Mentioned above

1. **Updating Packages**

We can update the package which we already installed by using this code “npm install <package name>@version”

1. **Using installed Packages**

* We commonly use commonJS or ESM to import and use the packages
* The default way to use modules in node.js is called commonJS using the function require(id)
* const chalk = require('chalk');
* another approach is called ES6 import using the keyword “import” instead of a require(id)
* import {red, blue} from 'chalk';

1. **Running Scripts**

* We can create the custom script name and can use like “npm run <custom script name>”
* Like “npm run start, npm run initialize (own name)”

1. **npm workspace (advanced)**
2. **Private Packages(advanced)**
3. **Public Packages (advanced)**

# **Error Handling**

The way to fix the bugs as early as humanly possible

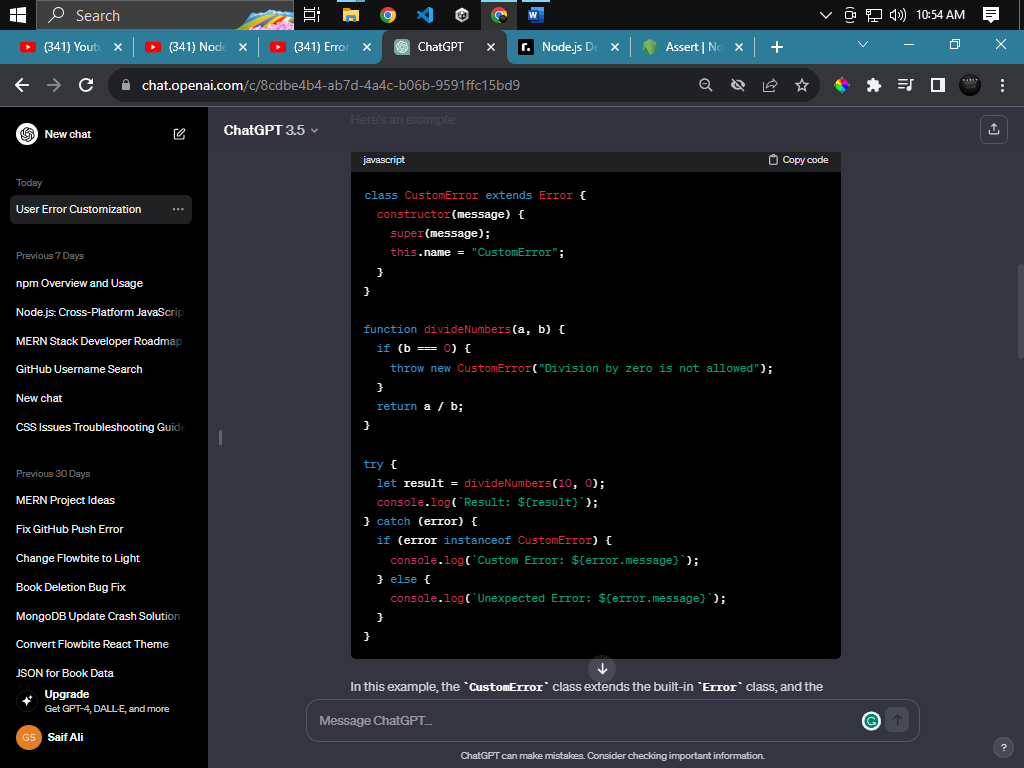
Error in node can be either operation or program errors

1. **Uncaught Exceptions**

* When an error is not handled properly than an uncaughtException is emitted
* This is done when programmer made an error

1. **Call Stacks and Stack Trace**
2. **Types of Errors**
   * 1. JS Errors

* JS errors made by JS to inform developers
  + 1. System Errors
* Node.js generates the system errors when an exception occurs within the runtime environment
* It is usually when an application violated the OS constraints
* Like try to read the file which doesn’t exist
* Examples are
  + EACCES - Permission denied
  + EADDRINUSE - Address already in use
  + ECONNRESET - Connection reset by peer
  + EEXIST - File exists
  + EISDIR - Is a directory
  + EMFILE - Too many open files in system
  + ENOENT - No such file or directory
  + ENOTDIR - Not a directory
  + ENOTEMPTY - Directory not empty
  + ENOTFOUND - DNS lookup failed
  + EPERM - Operation not permitted
  + EPIPE - Broken Pipe
  + ETIMEDOUT - Operation timed out
    1. User Specified Errors
* The way to handle the error using try catch
* We use the error object to show the detail about the error
* Error object has the property name, status, stack etc.

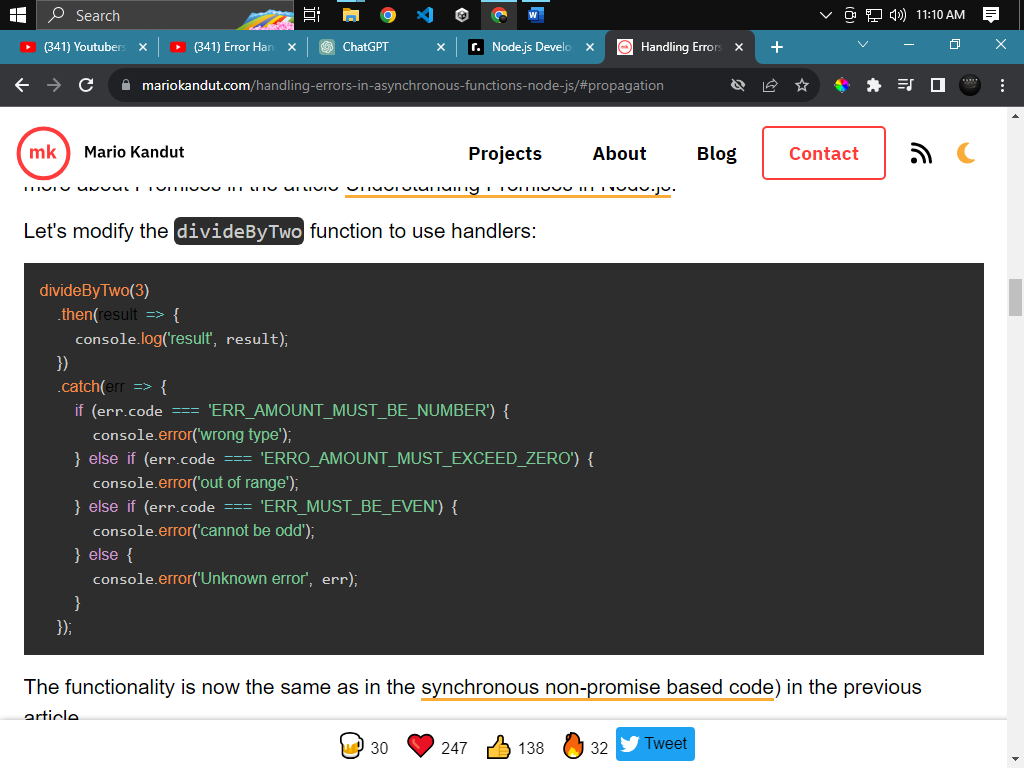


1. **Assertion Error**

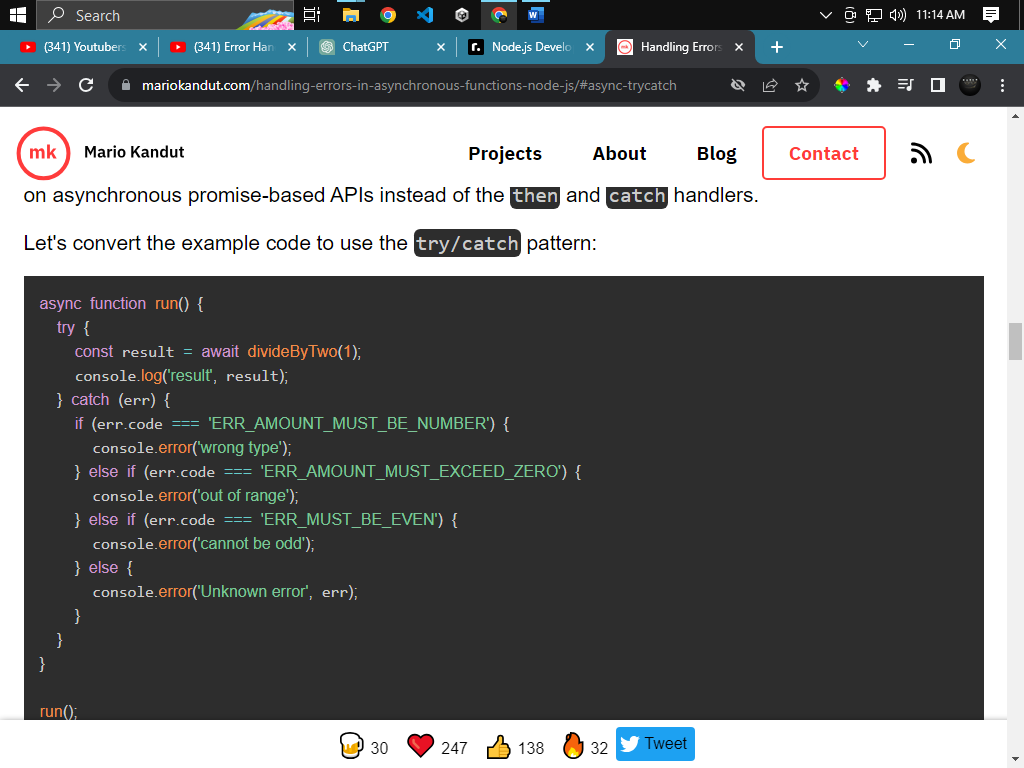
* An assert is the module
* An assertionError in Node.js through the error when the given expression in not truthy
* Like assert (a===b,” a is not equal to b”)
* It will run when condition is false

1. **Handling Async Errors (Intermediate)**

* When an error occurs in sync function it is called as exception but the error occurs in Promise, called asynchronous error or a promise rejection
* Asynchronous errors cannot be handled by using try catch
* It can only be handled inside the callback functions
* There are three ways to resolve the async error
* **Rejection**
* This error can be resolve in the promise rejection
* We use .than .catch handler to handle the errors



* **Try catch**
* We use try catch instead of then catch
* It is used promise-bases API
* Function should be returned with async and called with await



* **Propagation**

# **Asynchronous Programming**

1. **Promises (intermediate)**

Promise is a function that return an object an object through which we can attach callback

Callback will wait until the promise is fulfilled or rejected

1. **Async/await (intermediate)**

Special syntax to work with promise with fashion

Add keyword before function that return a promise

1. **Callbacks (intermediate)**

A function is called as parameter

Function myname (callback) {…}

1. **setTimeout (intermediate)**

run the function after specific time of period

1. **setInterval (intermediate)**

repeat the task (function) after some time of interval

1. **setImmediate (intermediate)**

it is Deprecated and non-standard

run the specific callback immediately after block all operation of running code

1. **process.nextTick (intermediate)**
2. **event emitter**

it is a class, not a function, with CamelCase

use to trig the event multiple times or once

1. **event loop (advanced)**

# **working With Files:**

1. **fs module**

use to interact with file system using JS code

all file operations have sync, callback, and promise-based forms

they all can be accessed using commonJS and ES6 Modules

1. **path module**

built-in node.js core module

const path = require('node:path');

use for working with files and directory paths

1. **process.cwd()**

returns the current working directory of the Node.js process.

1. **\_\_dirname**

Returns the path of the folder where JS file resides

Give the name of the current executing file

1. **\_\_filename**

Same as \_\_dirname but give the file name

1. **Glob**

Commonly used to specify filename and string

Glob patterns can only contain forward-slashes, not backward-slashes

1. **Globby**

Same as Glob with some useful features

1. **Fs-extra**

Same as fs module with some extra methods and supporting promise

It prevents EMFILE error

Alternate of fs module

1. **Chokidar**

Give it bunch of files and it watch them and notify every time when old file is edited and new file is created

# **Command Line Apps**

They are also called CLI (command line interface) application because they can run from the command line

They are useful for automation and building tools

1. **Process.env**

A global variable that is initialized through runtime and can be accessed everywhere

1. **Dotenv package**

Dotenv is zero dependency module that loads environmental variables from .env to process.env

1. **Environment Variables**

Variables set outside the code and cab accessed by process.env

They are not globally accessible and they are session-specific

1. **Existing and Exit Codes**

Terminate the node.js process by using process module

1. **Printing Output**
2. **Taking Input**

Node.js provides ways to take inputs from the user with an intuitive interface.

1. **Command Line Args**

# **Working With APIs**

API allows two app to communicate with each other

1. **http module**

* built -in node.js module
* transfer data over http
* it creates http server that listens to server port and give response to the client

1. **express.js framework**

it provides broad features to develop web and mobile app.

For single page, multi-page, and hybrid web apps.

1. **nestjs framework**

node.js framework to create efficient and scalable server-side app

1. **fastify framework**
2. **Making API calls**
3. **Authentication**

JWT: JSON Web Token is a popular method for authentication in node.js app to transmit infor between parties as a JSON object.

Token is generated on server-side and send to the client if the credentials are correct

It isn’t a middleware of node.js

“npm install jsonwebtoken express”

Passport.js: Passport is a popular authentication middleware for Node.js.

“npm install express passport passport-local express-session”

# **Keeping Application Running**

1. **Nodemon package**

Automatic restart the app

Use nodemon instead of node to start the project

# **Templating Engines**

Actually, the HTML but we can use it with dynamic content like we use in JS ${variable}

1. **Pug**

Same as mentioned above. Widely used

1. **Ejs**

Generate HTML markup and can use like pure JS

1. **Marko**

Fast and light weight

Compile templated to CJS modules and support streaming, async rendering and custom tags

1. **Handlebars**

Most popular like HTML (prefared)

# **Working With Database**

Collection of structured info and managed by DBMS

1. **Mongoose package**

* Is a popular ODM (Object Data Modelling) library of mongo and node
* Provides way to interact with MongoDB by defining schema, models, and relationship between data

1. **Prisma package**

Next generation ORM in typescript ecosystem

1. **Knex package**
2. **typeORM package**
3. **sequelize package**
4. **native driver**

to connect with MongoDB, we need a MongoDB node.js Driver

# **Testing**

Process of verifying

1. **Jest**

It is JS testing framework, focus on simplicity

It works with babel, typescript, Node, React, Angular, Vue and more

1. **Mocha**
2. **Cypress**

# **Logging**

Log actually help us to develop the application with less error and bugs

It shows the errors, warns, info and debug like console in browser

1. **Winston**
2. **Margon**

# **Keeping App Running (intermediate)**

1. **Pm2 package**

Keep app running even we close the terminal

1. **Forever package**
2. **Nohup**

# **Threads (advanced)**

1. **Child process**
2. **Cluster**
3. **Worker threads**

# **Streams (advanced)**

# **More Debugging (advanced)**

1. **Garbage Collection..**
2. **Memory leaks..**
3. **Node-inspect**
4. **Using APM**

# **Common Build-in Modules (intermediate)**

1. Fs: Dealing with file system
2. Os: give info about the system
3. Net: build clients and servers
4. Path: handle file paths
5. Url: help in parsing url strings
6. Events: handle existed events and create custom events
7. http: transfer data over http
8. console: log info in the console
9. assert: provides set of assertion tests
10. process: provide info and control the current process
11. cluster: create the child process over the same port that runs simultaneously
12. perf\_hooks: provide API for performance measurement
13. crypto: to handle cryptographic functions
14. buffer: provides API to handle stream of binary data