**NodeJS**

What is NodeJS?

* It’s a **runtime environment** that allows you to execute JS code outside of web browser.
* It’s built upon **V8 JavaScript engine** same as chrome.
* **Handles asynchronous operations**.
* **Non-blocking I/O models**, that allows efficient handling of multiple tasks simultaneously.

Key Features:

* **Single Threaded**: NodeJS is single threaded which means it’ll handle all request in a single thread.
* **Event Loop**: NodeJS uses even loop to handle asynchronous tasks. Once this task is in process it delegates current tasks to underlying system (It can be multiple thread or other system) and continues with other tasks and once the previous task is completed it receives a callback and processes it.

Used For:

* APIs obviously
* Server-rendered-apps
* Microservices, etc

Disadvantage:

* Not good for CPU intensive work

Modules Import / Export:

* While exporting:
  + module.export = funct() // we can export anything object, array, ..
* While importing:
  + const func = require(‘./utils’)
  + console.log(func)

ES Module:

* In package.json, add ***type: “module”***
* Now, we can export import usually as we do in JS not like in just above Import/export.

Nodemon:

* Watches and restarts server.
* For specific files only like js, mjs, etc
* Change start in package.json from “node server” to “nodemon server”

.env

* Now NodeJS gives env support by default.
* We’ve to change “nodemon server” to “nodemon –env-file=.env server.js”

Event Emitter

* It’s a core NodeJS module, that is used to handle asynchronous events.
* It’s part of event module that is used to create, manage and respond to events in application.
* **Importing and creating an instance so we can use it:**
  + const EventEmitter = require('events');
  + const myEmitter = new EventEmitter();
* **Listening/Register event:**
  + myEmitter.on('event', () => {
  + console.log('An event occurred!');
  + });
* **Emitting the event:**
  + myEmitter.emit('event'); // This will log 'An event occurred!'
* **Passing the arguments:**
  + myEmitter.on('eventWithArgs', (arg1, arg2) => {
  + console.log(`Event received with arguments: ${arg1}, ${arg2}`);
  + });
  + myEmitter.emit('eventWithArgs', 'argument1', 'argument2');
  + // Logs: Event received with arguments: argument1, argument2
* **Using once:**
  + myEmitter.once('onceEvent', () => {
  + console.log('This event will be listened to only once.');
  + });
  + myEmitter.emit('onceEvent'); // This will log the message.
  + myEmitter.emit('onceEvent'); // This will not log anything.
* **Removing listeners:**
  + const callback = () => console.log('Event triggered');
  + myEmitter.on('removeEvent', callback);
  + myEmitter.removeListener('removeEvent', callback);
  + myEmitter.removeAllListeners('event'); // Remove all listeners

**ExpressJS**

What is this?

* It’s a **minimal** and **flexible** NodeJS framework for web application.
* It’s used to build web applications and APIs quickly and easily.
* It’s **unopinionated.** (It means it doesn’t have any structure to follow live Django we can structure it anyway we want.)

Router:

userRouter.js

const express = require('express');

const router = express.Router();

// Get all users

router.get('/', (req, res) => {

res.send('User List');

});

// Create a new user

router.post('/', (req, res) => {

res.send('Create User');

});

// Get a specific user by ID

router.get('/:id', (req, res) => {

res.send(`Get User with ID: ${req.params.id}`);

});

module.exports = router;

server.js

const express = require('express');

const app = express();

const userRoutes = require('./userRoutes');

// Use the user routes for paths starting with /users

app.use('/users', userRoutes);

app.listen(3000, () => {

console.log('Server is running on http://localhost:3000');

});

Custom Error Handler

* It’s a middleware provided by expressJS that handles error obviously in req-res cycle.
* Example:
  + function errorHandler(err, req, res, next) {
  + console.error(err.stack); // Log the error stack trace
  + res.status(500).json({ message: 'Something went wrong!', error: err.message });
  + }
  + We just have to attach this in main file to use this
  + // Attach the custom error handler as the last middleware app.use(errorHandler);

e