GUI-

* Graphical User Interface that helps user to graphically interact with the app.
* Communication between user and app.
* Eg:-Icons on the screen that helps user to interact with app. (Facebbok.com)

API-

* Application Programming Interface is an interface(software) that connect/communicates between two apps.
* Eg:-Travel Website, login with google,paypal in ecommerce website.
* Eg:-Weather App shows map data of Google maps through Google Map API Key. [Google map API helps to interact between Weather APP and Google Map.]
* Eg:- Waiter establish communication between customer and chef.

REST(An style to perform API by transfer data in many formats)-

* Representational State Transfer is an architectural style for API that uses HTTP requests to perform CRUD.
* We will be having same URL for CRUD.
* We are transferring the state of a resource when requested.6 constraints.
* State of resource at any particular time is resource representation.
* Only using mongoose then we will have different URLs for CRUD.

require()=

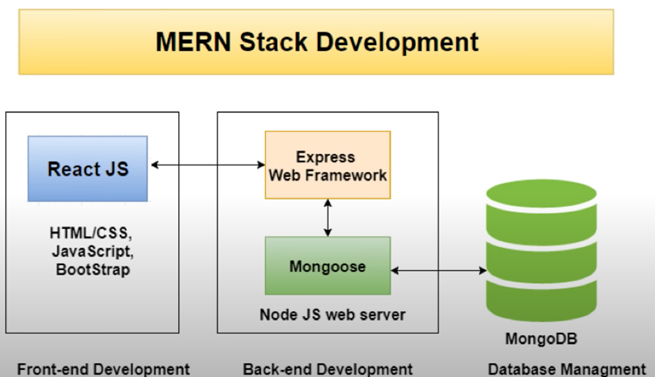
* -it reads a JavaScript file
* -executes the file
* -return the exports object
* var example = require('./sample.js')
* example = {
* message: "hi",
* say: [Function]
* }

Module.exports ~ exports-

* Instructions that tell Node.js which bits of code (functions, objects, strings, etc.) to export from a given file so that other files are allowed to access the exported code.

express.Router()  =

* When var app = express() is called, an app object is returned. Think of this as the main app.
* When var router = express.Router() is called, a slightly different mini app is returned.
* The idea behind the mini app is that each route in your app can become quite complicated, and you'd benefit from moving all that code into a separate file.
* Each file's router becomes a mini app, which has a very similar structure to the main app.



# Server-serve data/service to everywhere

Environment Variables = For Global(environment) use to run on any environment(Local+Cloud). Different environment (e.g.; staging, testing, production).

In node.js, environment variables are accessed using env object(Process object property). Process{

env:{ …

}

…}

Process = Global object in Node.js that can be accessed inside any module without requiring. It provides various information about the runtime of a program.

It is an instance of [EventEmitter](https://nodejs.org/api/events.html#class-eventemitter).

Dotenv = npm package that automatically loads environment variables from a .env file into the process.env object.

To connect our own created environment variable in .env file to process.env object.

Process.on(eventName,listener(callback)) = Adds the listener function to the end of the listeners array for the event named eventName.

process.on("uncaughtException", (err) => {})

Uncaught Exceptions = Exceptions that are not caught. (EG:-Variable used but not created).

err [<Error>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Error) The uncaught exception.

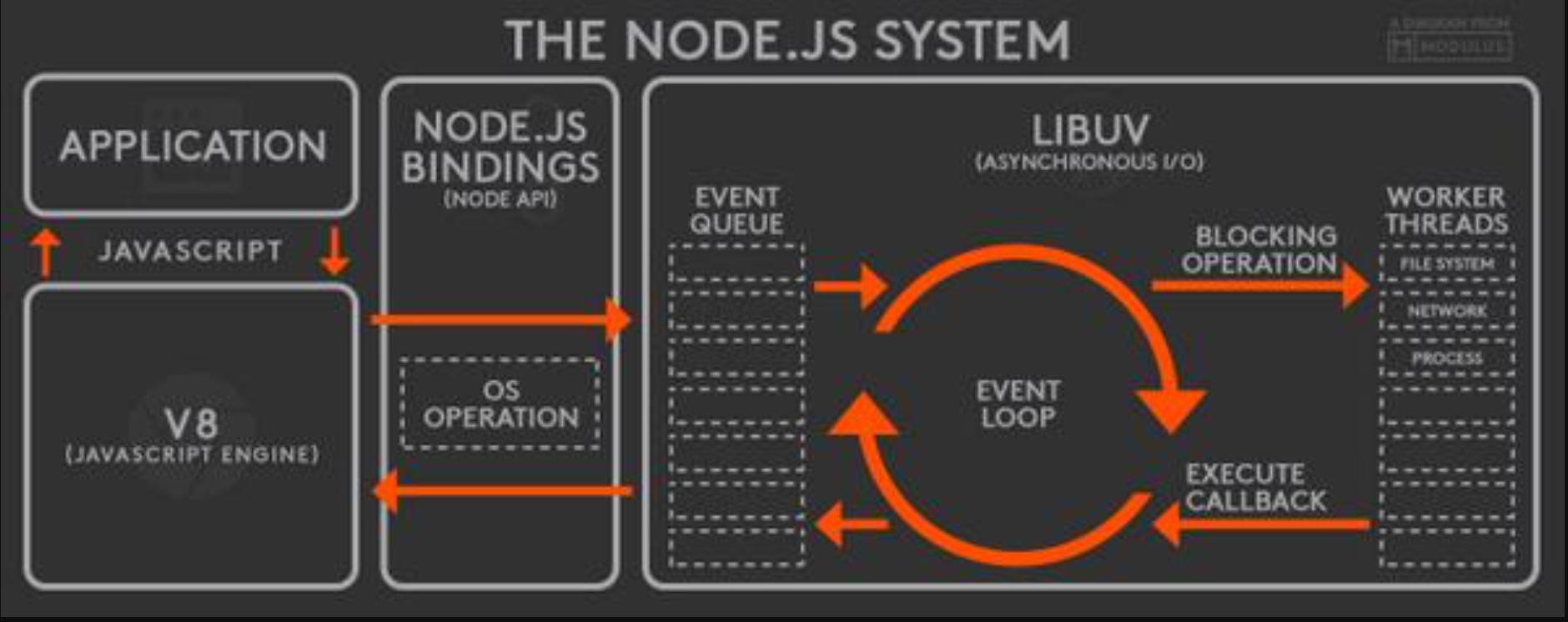
process. exit(1) = To exit the application. Failure Code=1, Normal Code=0

unhandled Rejection = event is emitted whenever a Promise is rejected and no error handler is attached to the promise

# app.listen() = binds(client and server) and listen to the connections(requests) of client on specified host and port.

# Its Prepare the server (start the server)(Executes the server)(make it live.) on that port number and host.

# Now, when client comes to that port and host, it finds that app running. That is, its listening to incoming client requests.

Node.js = This is the environment. 

app.use() = a middleware function with no mount path. This code is executed for every request to the router.

Middleware =

* The routing methods specify a callback function (sometimes called “handler functions”) called when the application receives a request to the specified route (endpoint) and HTTP method(GET,POST,PUT).
* application “listens” for requests that match the specified route(s)(Endpoint/path) and method(s) (GET,POST,PUT)., and when it detects a match, it calls the specified callback function.
* In fact, the routing methods can have more than one callback function as arguments. With multiple callback functions, it is important to provide next as an argument to the callback function and then call next() within the body of the function to hand off control to the next callback.
* You NEED  express.json() and express.urlencoded() for POST and PUT requests, because in both these requests you are sending data (in the form of some data object) to the server and you are asking the server to accept or store that data (object), which is enclosed in the body (i.e. req.body) of that (POST or PUT) Request.

express.json() is a body parser for post request except html post form [converts request body to JSON ]

 express.urlencoded({extended: false}) is a body parser for html post form.[ converts request body to JSON, converting form-data to JSON ]

mongoose =

Express Web Framework (Frame+Work)

EG:- a ‘Photo frame’ in which a photo must be of shape and size that of frame to work.

EG:- Brick Frame,House Frame(Everything will built inside/according to that frame)

Web Framework means you have to code inside a skeleton that is predefined.

Arrangements in which software provides greater functionality that can be extended by additional user written code.

Advantages:-

* Allow standard way of creating applications.
* Have functions resuable.
* Simply as the process of creating web app-basic is already been created.(Not scratch).
* Hiding startup things(focus on main part)
* A way of processing requests(MVC Model)(Just logic not to understand flow).
* Have updated features.
* Attach great third party.

express() = By require we get the whole express. But to use its function,etc., we need to initialize/ start express as it’s our whole application(+Framework).

* **Routing** refers to how an application’s endpoints (URIs) respond to client requests.

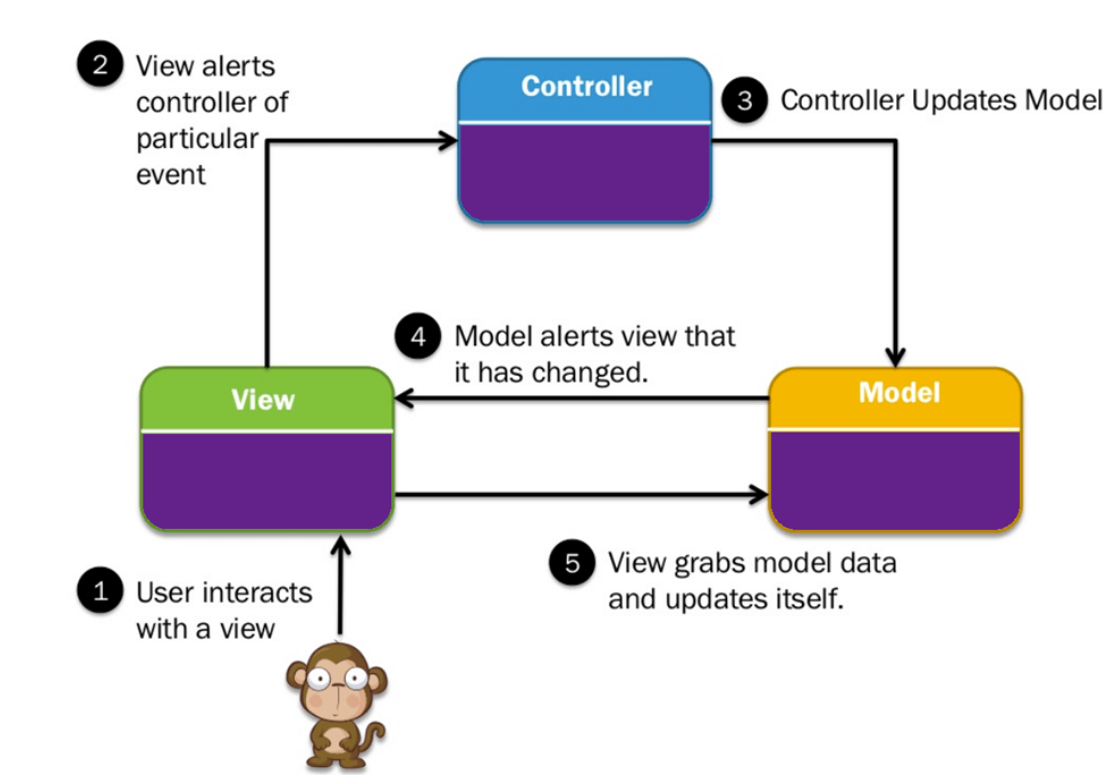
express.json() =

* It parses incoming requests with JSON payloads(content-type-to make request headers) and is based on body-parser.
* It parses arguments from an incoming request and uses them as inputs to invoke the corresponding controller method

MVC

* Model: It includes all the data and its related logic
* View: Present data to the user or handles user interaction
* Controller: An interface between Model and View components





Model = Schema Database

View = React

Controller = Express

Query String = is a part of a [uniform resource locator](https://en.wikipedia.org/wiki/Uniform_resource_locator) (URL) that assigns values to specified fields(key).

A query string commonly includes fields added to a base URL by a Web browser or other client application.EG:- https://example.com/over/there?name=ferret

The query string is name=ferret