**Setup**:

* **Introduction**:
* This setup is for backend and implemented in NodeJS.
* This setup provides various commands which helps developer to build folder structure, apis, middlewares, database configuration, etc… easily.
* This setup provides connection with MySQL database using sequelize.
* It provides error handler so that, whenever error generates, app doesn’t get crashed. It will show the error, its reason and when error generated, along with that it will also show the line number and file path from where error generates.
* If defined port is busy then it asks to switch to the next nearer free port.
* It provides file upload solution.
* It provides the feature for api documentation.
* With ‘npm start’ command, server starts when it calls and using ‘npm run dev’ command, server starts automatically.
* **Folder Structure:**
* api
* api folder will have modules that will be created by developers, and in those modules, there will be controllers, middlewares and services folders and routes.json file.
* moduleName

->In big projects, there may many routes and controllers and middlewares used. So to manage all the actions clearly, we should separate the things in modules. Ex. In E-Commerce website, there are many modules like shipping, payment, admin panel, user profile, products, etc.. And each modules will have many routes like in shipping - remainingShipment, completedShipment, In products – addProduct, deleteProduct, editProduct, etc… So to manage all these routes and their actions and HTTP methods, we will separate these fields as a module.

* controllers

->controllers folder will have controller files created by developers and the apis will be in object (key-value) format. For handling the error in api, it has setup.findErr(err) function, Developer just needs to pass error to that function and it will show the error information.

Ex. module.exports = {

Api: () => {}

}

* middlewares

->middlewares folder will have middleware files used in apis that are declared in controller files, created by developers and the middleware is also in object (key-value) format.

Ex. module.exports = {

middleware: () => {}

}

* services

->services folder will have service files used in apis that are declared in controller files, created by developers and the service is also in object (key-value) format. There is a global object named ‘setup’.

Ex. module.exports = {

service: () => {}

}

To import service in api from service folder, we need to use setup.services[‘folderName’][‘fileName’]

* routes.json

->As name suggests, it defines routes. It has json data in which all the endpoints, HTTP method of each endpoint, what action it needs to take place when calling the particular endpoint, which middlewares particular endpoint will use, the endpoint is publicly accessed or not, endpoint should be called from root or not (here root will be the moduleName). These all informations will be saved in json object of json array.

Ex: [

{

path: ‘{endpoint}’,

method: ‘{HTTP metho}’,

action: ‘FileName.FunctionName’,

middleware: [‘FileName.FunctionName’],

globalMiddleware: [‘FileName.FunctionName’],

public: true/false,

root: true/false

}

]

* config
* config.js

->This file has configurations for showing the data of which url, HTTP method, http version, status is being called with current date and time in colsole. It uses ‘morgan’ package.

* config.json

->This file has json data in which developer can define there req-headers or req-body, where req-body defines what body is going to be passed when url is being called and req-headers defines the headers is used in calling the url, whatever developer wants to see in console when url executes, that developer needs to define there. Config.js will work on this data and it will show information in console.

* database.json

->This file has json data of required information like username, password, database name, host, dialect etc... For connection with database in development and production area.

* fileUpload.js

->This file has configuration for uploading any files to given location. This file uses ‘multer’ package. Developer just needs to write setup.store({LocationPath}) to store files and setup.uploadFile which is an object to define files.

* core
* connection.js

->This file establishes the database connection using sequelize. It will ask developer for DB logs with date and time whether developer wants to save logs in files or show logs in console or both.

* crons.js

-> This file runs all the crons files of crons folder created by developer. If any cron is scheduled off and at particular time, it should get on and off, for that developer needs to write setup.crons[‘FileName’][‘cronName’] to call a particular cron.

* functions.js

-> It has configuration for calling function files from function folder. Developer just needs to write setup.functions[‘folderName’][‘fileName’][‘functionName’]( params1,params2,…,paramsN) to call the function. Functions folder can have multiple folders and files.

* migrations.js

-> This file handles all the migrations. If any migration is pending, then it shows the pending migration files to developer and asks to run.

* routes.js

-> This file handles all the routes which are defined in routes.json and maps path to the api, middlewares, global middlewares which is declared in routes.json.

* services.js

-> This file handles the configurations of services. Developer can access the service files in api via setup.services[‘fileName’][‘serviceName’]

* crons

-> This folder can have multiple cron files created by developer. Crons will be in object (key-value pair) format. Ex.: module.exports={

cron: ()=>{}

}

* db
* migrations

-> This folder includes multiple migration files.

* models

-> This folder includes multiple model files.

* seeders

-> This folder includes seeder files.

* dbLogs

-> This folder includes database log files.

* functions

-> This folder holds multiple folders and multiple files.

* middlewares

-> It holds middleware files which are common for many apis or modules. These middlewares are global middlewares.

* src
* app.js

-> This is the root file where the process starts. It imports all the required files and route calling starts from here. It includes error handler also which handles all the errors so that app doesn’t get crashed. It has swagger configuration also for api documentation.

* uploads

-> This folder includes files uploaded by end user if developer has not set path for files storage.

* .env

-> This file includes port number defined by developer.

* .sequelizerc

-> This file defines the path of database folders and file like migrations, seeders, models and database.json.