# **NODE-API Setup:**

1. **Server Setup**
2. **Postman & Cors**
3. **Database setup with MongoDB-atlas**
4. **Database Setup Locally**
5. **Separating Routes**
6. **Controllers**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# 

# 

# 

# 

# 

# 

# 

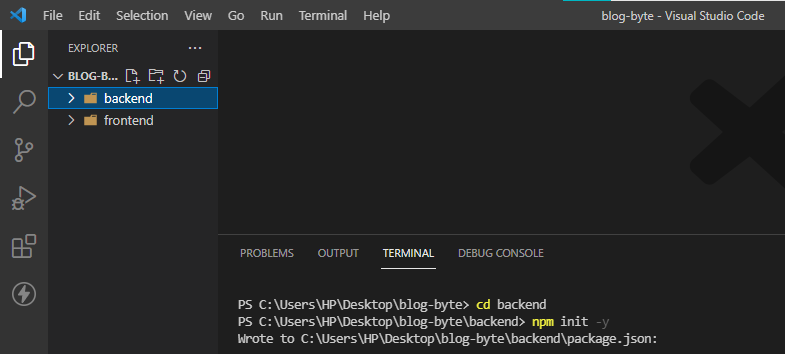
# **TASK-1 SERVER SETUP**

**STEPS :**

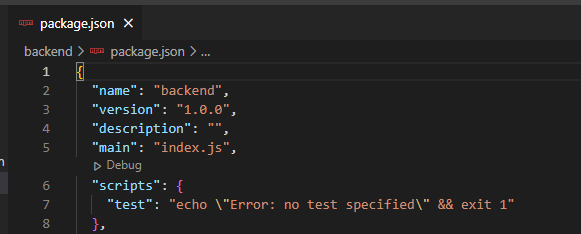
* Create main folder for the project => **blog-byte**
* Inside blog-byte create 2 folders=> **1. frontend 2. backend**

|  |
| --- |

* First we need to initialize our project with package.json file so that we can bring in NPM packages and build up our application.
* Open folder **backend** in Visual Studio Code.
* Open terminal and run command=> **npm init -y**  (-y means we are going to accept default that we get with this command)



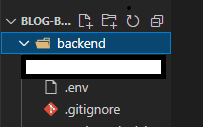
* This will create a **package.json** file in the backend folder.
* Some info will be generated after running command => name, description, version etc.



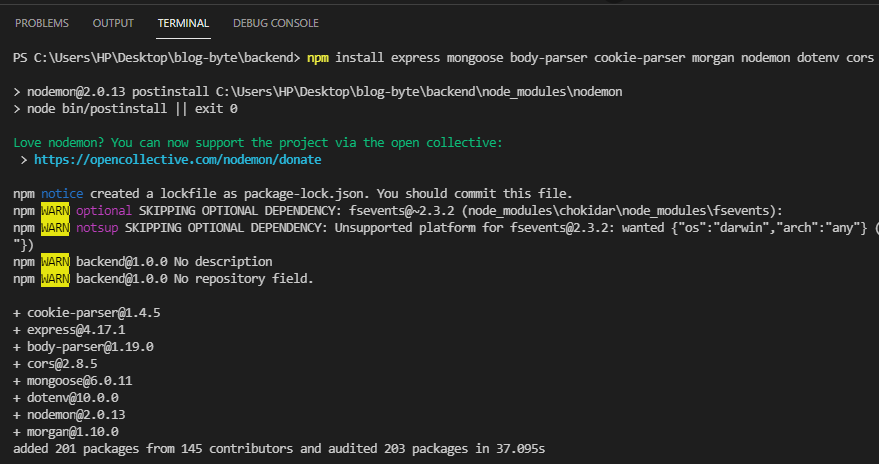
**=> What is Package-JSON ?**

The package. json file is the heart of any Node project. It records important metadata about a project which is required before publishing to NPM, and also defines functional attributes of a project that npm uses to install dependencies, run scripts, and identify the entry point to our package.

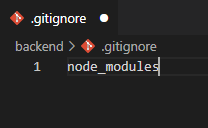
* Now in the **backend** folder create a **.gitignore** file.
* Create another file in **backend** folder named **.env** to store environment variables.



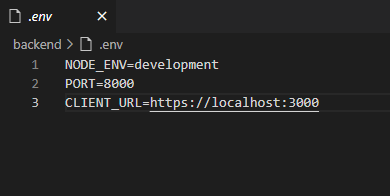
* Now run command => **npm install express mongoose body-parser cookie-parser morgan nodemon dotenv cors**
* First thing we need is Express and we need mongars to talk with mongar.
* These file will help to parse jason data from request body. We will be sending routers from frontend to backend.So this will give us jason data otherwise we’ll not see anything.
* Morgan will help us see the endpoints in terminal. So when some requests are coming from the client side to the back end you will see if it is coming to our local host blogs then you'll see that printed in the console which is pretty good in the development mode again.
* Nodemon so we need not one not one we can use to continuously run our node server again.
* Dotenv so we can get access to our environment variables using .env .
* Cors so our API is accessible to our front client, otherwise we’ll get errors like **cross-origin resource sharing.** The CORS mechanism supports secure cross-origin requests and data transfers between browsers and servers.

****

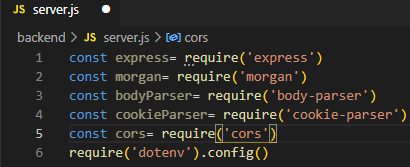
* Now open **.gitignore** file and add **node\_modules** folder in it.
* Node Modules => folder contain all the packages so we don't need to push them on GitHub when we push our project.



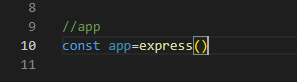
* Now open **.env** and write some env variables here. Write **NODE\_ENV=development** . So it is in development mode now. While pushing project to live server we’ll change this to **=production**.
* Specify the port as **PORT=8000**. So port 8000 will be used by our server.
* Specify **CLIENT\_URL=**[**https://localhost:3000**](https://localhost:3000). This will change when we’ll push the project to the live server.



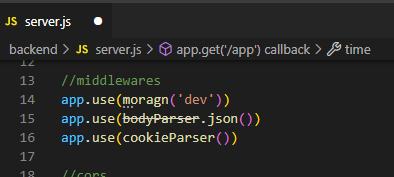
* Now lets build up our server. Create a new file **server.js** in the **backend** folder.
* Let's start bringing in packages in it.



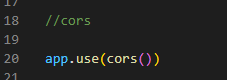
* Now let's create our **app** using express. And invoke express app by express().So express app will be available in app variable.



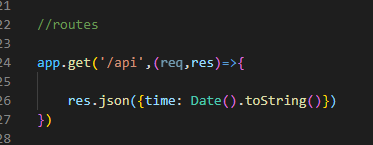
* App will use **middleware**. So we apply middlewares using the **use()** method. **mogan(‘dev’’)**=> means in development mode that will give endpoints in the console.



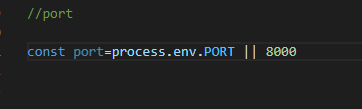
* Set **cors**.



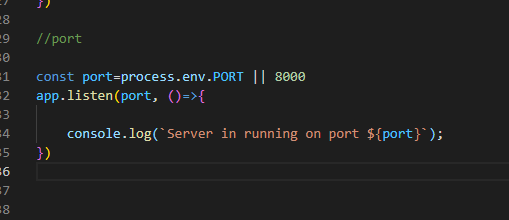
* Set **Routes**, that will handle the incoming request. So, anytime our server requests to slice API, our server responds current time in jason format.



* Now lets set **ports**, because we need to run server and port 8000. (port coming from env file=> we can get access to them using dotenv) And if it doesn't exist we can pass default value of 8000 using OR(||) operator.



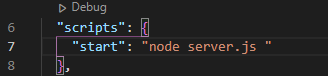
* Now we can run our app so the **app can listen on port.** Second argument to display post value in console using a template string.



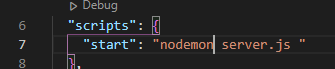
* This is our Node JS Server. Save this file. So this is the entry point for our server.
* Now we need to make a small change in **package.json** file to use npm start to run to app.



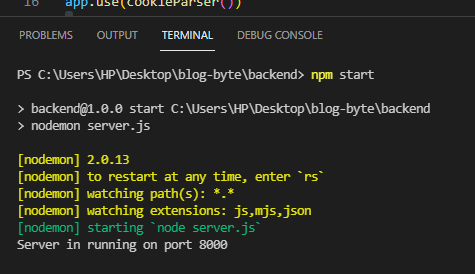
* Replace test with **start** and **node** with **server.js** file.



* But drawback of this is everytime we make changes, we have to stop the server, and start again. To avoid this, use **nodemon** instead of **node**. Nodemon will execute our file first time and anytime there is a change in the file. It will automatically restart the server.



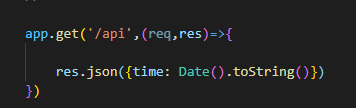
* Now we should be able to run **server.js**. Run the code, and get response and endpoint (that we set in routes).
* In the terminal move to **backend** folder and run **npm start** command. It will show server is running on **port 8000**.



* Now open the browser and run **localhost:8000**. It will not return anything because we haven't handled the default route.

|  |
| --- |

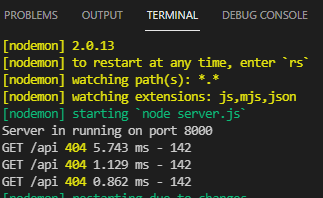
* However we are handling api endpoint. So we can run **localhost:8000/api .** We will see current time, because we are responding with current time when there is a request in **/api**.



* Our server is running in the browser we can access it by **localhost:8000/api**.

|  |
| --- |

* Now we can see **endpoints** in the terminal.(because of morgan).



**=> Our node js server is running now, Task Completed… :)**

# **TASK-2 POSTMAN & CORS**

* **Postman:**

Postman is an application used for API testing. It is an HTTP client that tests HTTP requests, utilizing a graphical user interface, through which we obtain different types of responses that need to be subsequently validated.

* **CORS:**

Cross-origin resource sharing is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served. A web page may freely embed cross-origin images, stylesheets, scripts, iframes, and videos.

**STEPS :**

* Now we will use a tool called **postman** to test our **API**. First we have to download postman.

|  |
| --- |

* After downloading, install the postman’s setup.

|  |
| --- |

* Now make a new account on postman.

|  |
| --- |

* <https://app.getpostman.com/join-team?invite_code=9222a7834efe1303fc08cb9a29319ee0>
* After creating account, open Postman on you Pc and sign-in to your account.

|  |
| --- |

* Create a new workspace. We created on with name **Blog-Byte**.

|  |
| --- |

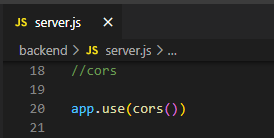
* From postman we can make request to our API. To see request make sure you have **GET** request.
* Type URL **localhost:8000/api.** Now send request by **send** button.

|  |
| --- |

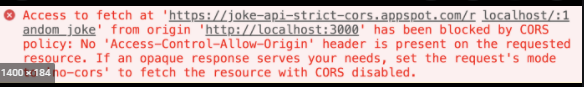
* We can see the requested response in the window below. Current time is displayed which we requested from our api.

|  |
| --- |

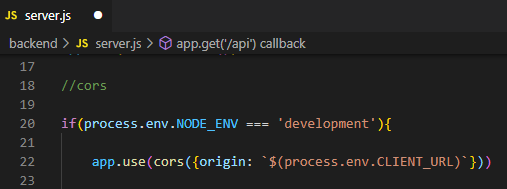
* There’s one thing more we can do with our server. We have applied **cors,** But we haven't passed any configuration.



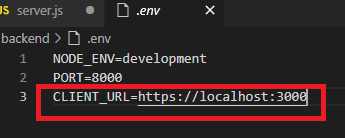
* In development ,we’ll be running our backend in **port 8000,** which is a different origin. But our client will be running in **port 3000** which is of a different origin.
* So, when we make a request from client side to our server side we’ll see an **error** in the browsers, called **CORS error** (because our API is running in different port & our client is running in different port).



* Browsers by default don't allow such access because of security reasons.
* To avoid that during development, we apply **CORS**. Write the following check code to apply cors.



* **CLIENT\_URL** refers to following.



* So, when we are developing locally, we’ll allow access to this origin. *This works only between browsers to browsers communication. So, does not have any affect on postman. Postman can still be used.*

**=> Postman & cors has been setup successfully,**

**Task Completed… :)**

# **TASK-3**

# **DATABASE SETUP WITH MONGODB-atlas**

* Now its time to connect our node js API with mongoDB, and there are 2 options:

1. Use online service like **mongoDB-atlas**
2. Installing **mongoDB** locally in your computer (prefered)

* **Setting up online MongoDB-atlas**
* To use online, create a new account on **mongoDB-atlas.** Now, MongoDb-atlas is set to use.
* Create a new cluster.

|  |
| --- |

* Choose aws from cloud providers.

|  |
| --- |

* Now select **storage space**, **cluster name** and **create cluster** .

|  |
| --- |

* The cluster named **blogbyte** have been created in mongoDB-atlas.

|  |
| --- |

* Always use this link to login in to mongoDB-atlas <https://cloud.mongodb.com/v2/6173b687f1f1fe3e412d4988#clusters?fastPoll=true>

|  |
| --- |

* Set up the 2 fields given below.

|  |
| --- |

|  |
| --- |

|  |
| --- |

|  |
| --- |

|  |
| --- |

* After setting up the above fields, choose a connection method.

|  |
| --- |

* Choose connection method as, **connect you application**.

|  |
| --- |

* Copy the link of connection string given below.

|  |
| --- |

* Open Project, go to **.env file**. Create a new environment variable here. Like

**DATABASE=**’paste MongoDB-atlas link here’

|  |
| --- |

* After pasting link, change **<password>**  field with your DB password and **myFirstDatabase** with your database name. (If you password have special characters use **percent-encoding** while writing it in connection string.

In our case=> **blogbyte@4 = blogbyte%404**)

|  |
| --- |

* Now we will be able to use MongoDB-atlas as our database.
* Let's go to **server.js** file and connect to the database.

|  |
| --- |

* Mongoose official docs. <https://mongoosejs.com/>. Use mogoose .connect method given below to connect to database.
* To access environment variables from .env file use (**process.env.varName**)

mongoose

.connect(process.env.DATABASE, {

useNewUrlParser: true,

useUnifiedTopology: true

}

|  |
| --- |

* That's how we can use mongoDB using the **mongoose .connect()** method. And this gives us the **promise.**
* We can access promise by using **.then()** method. So, display connection success in console in then().

|  |
| --- |

* Now, start the server and see the console for your app running info. Console will show our msg **“Connected to MongoDB!!!”**
* (if some other app is running on the same port, use **killall node** or **pkill 8000** if you want to stop the process running on port 8000. Then start server again using **npm start**).

|  |
| --- |

* Here’s our console displaying success msg.
* So,we have successfully connected our application with mongoDB-atlas that is running in the cloud.

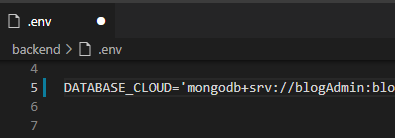
**=>MongoDB-atlas setup successfully, Task Completed… :)**

# **TASK-4**

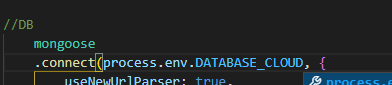
# **DATABASE-SETUP LOCALLY**

**STEPS :**

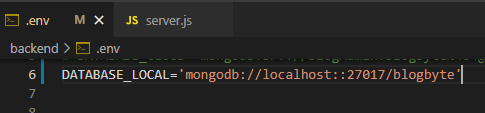
* Go to **.env file,** rename **DATABASE** to **DATABASE\_CLOUD.**



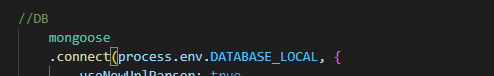
* Go to **server.js file,** rename **DATABASE** to **DATABASE\_CLOUD.**



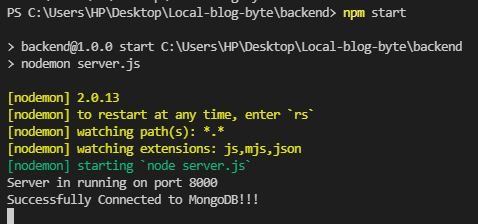
* Restart server and check if it is working fine.
* Go to **.env file,** rename **DATABASE** to **DATABASE\_LOCAL.**



* Go to **server.js file,** rename **DATABASE** to **DATABASE\_LOCAL.**



* Restart server and check if it is working fine.



**=>Database connection has been locally setup successfully,**

**Task Completed… :)**

## **Setting up MongoDB locally**

* Use **robo3T** (is MongoDB GUI with embedded shell)that will allow you to visually see your database.
* Download **robo3T** from the website as shown below.

|  |
| --- |

* After downloading, install the setup on your Pc.

|  |
| --- |

* After downloading, install the setup on your Pc. **Robo3T** is all set to use.

|  |
| --- |

* Now, download mongoDB locally on your Pc.
* Steps on link below:

<https://medium.com/@LondonAppBrewery/how-to-download-install-mongodb-on-windows-4ee4b3493514>

|  |
| --- |

* Now install MongoDB setup.

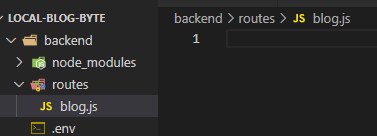
|  |
| --- |

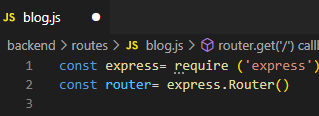
**steps => from git bash**

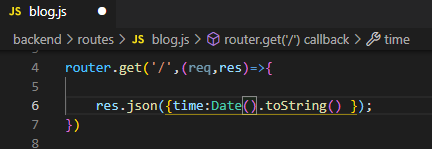
# **TASK-5 SEPARATING ROUTES**

**STEPS:**

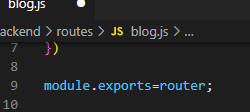
* To structure our project, we have to separate the routes.So,we are moving our routes to a separate routes folder.
* In **backend** folder, make a new folder named **routes**.
* Inside the **routes** folder, create a new file named **blog.js**. Because our application is a blogging application, we need a blogs route.



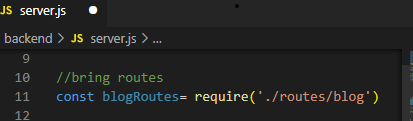
* We have to mention **express, express router.**
* We have to **handle incoming routes**, if we get a request on forward slash, then we can use a call back function and here we are responding with time.



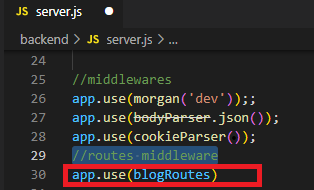
* Now we need to use this in our **server.js** file.So, we need to export routes.
* Any route we will use can be exported by using **module.export=router** that means add this routing to export object which is in the route of the node process.



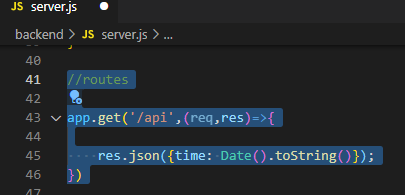
* Now we are going to import this in **server.js**.



* Now we can use this as middleware.We can use this **app.use()** method to apply in middleware.



* We can apply these routes to middleware to our express app.
* Remove these routes, because we already moved it to its own routes folder.



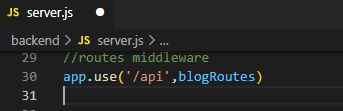
* Now restart the server, and check if it is working fine. Then open the browser and type URL **localhost:8000**, it will give time in response.

|  |
| --- |

* **Because we are building API, we need to have all our endpoints starting with /api**.

|  |
| --- |

* Give **/api** as first argument. So in this way any routes we define will have /api prefixed.



* Now **localhost:8000/** will give an error. Because request is handled at **localhost:8000/api.**

|  |
| --- |

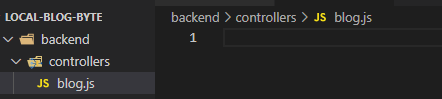
**=> ROUTES have been separated successfully,**

**Task Completed… :)**

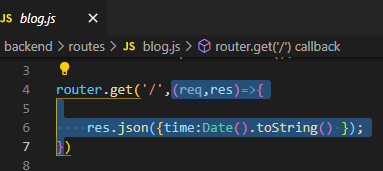
# **TASK-6 CONTROLLERS**

**STEPS:**

* Passing functionality of **routes** to **controllers** for handling. Because later after we have to deal with a lot of work with routes. (Need to go to database, create blogs,and associate with users).
* Create a new folder named **controllers** in **backend** folder and make a file named **blog.js**.

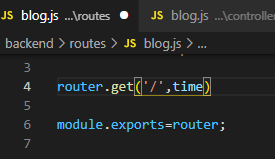


* Moving this functionality from routes to controllers.Remove it from here.





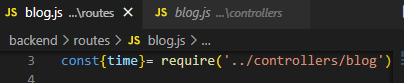
* Give the controller a name as given here as **time**.Any request coming to this endpoint will be handled by time controller method.



* Write **exports** and append this to an empty object. Now place here the control function that we removed from routes here, add it to export object.



* Now import it in **routes** **blog.js** file.



* Restart the server, and check if it is working fine.

|  |
| --- |

**=>Controllers have been setup successfully,**

**Task Completed… :)**