**Module 3 Backend to Database**

**Setting up the db:**

* Create the database **my\_tasklist** in the **Atlas or mlab** create the **user name as :** <your\_name> and **password as: 1234** (Don’t create the collection, it has to be created from the server side code).
* Copy the connection string (URI of the DB) after you have created the db (To be used later in express.js)

**Setting up the project:**

* Create a project in VSCode editor, Create package.json file and add the required dependencies **express, cors, body-parser, nodemon, mongodb driver dependencies.**

**Setting up the middleware:**

* Create index.js under server folder (server/index.js).
* Setup index.js as the default file for node and nodemon in the package.json file.
* Set up and listen to port 5000 in the index.js file.

**Creating the API methods:**

* Route the request for the URL: api/taskslist to tasklist.js from the middleware. (Research and write what is middle ware and write in the docx file mentioned in the requirements)
* In tasklist.js create an async loadTasksCollection() that connect with the database using String URL that you have copied after creating the database, create a collection mytasks and return.
* Create the router.get() that reads the mytasks collection and return all the tasks. (Routed for the url ‘/’ ).
* Create the router.post() that inserts a new task to the mytasks collection and return the status code 201 if successfully inserted (Fields to be inserted: **task** and **dateCreated**). (Routed for the url ‘/’ ).
* Create the router.delete() that deletes an existing task from the mytasks collection and return the status code 200 if successfully deleted. (Routed for the url ‘/<\_id of the task to be deleted>’).
* Use Postman (A software tool to send http requests – Very useful during real-time development) to test with the http request.

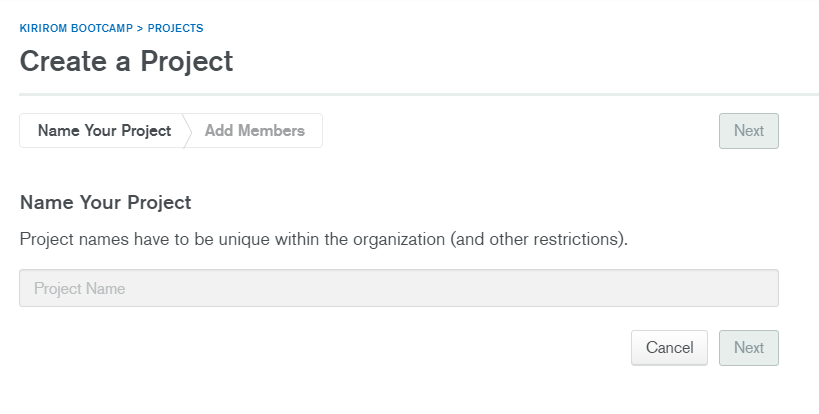
**Setting Up Step**

**Setting up the database:**

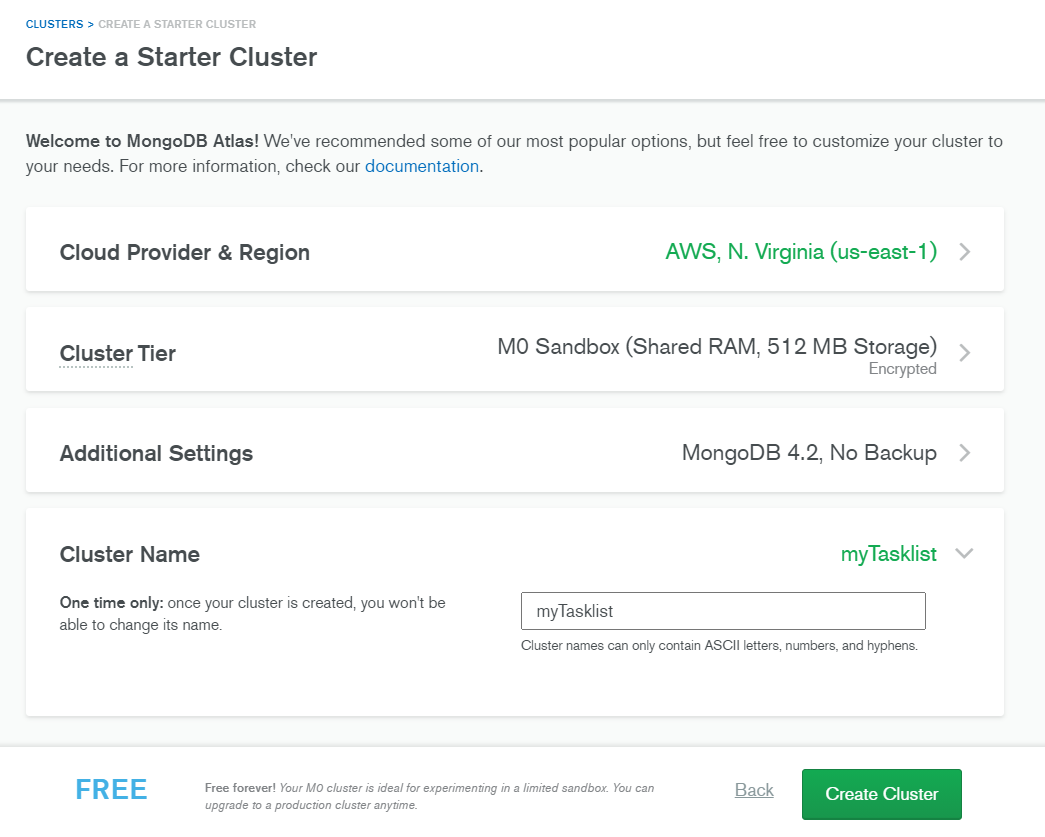
Step 1: Login to cloud mongodb:

<https://cloud.mongodb.com/v2#/org/5fb759989004e01a49288c10/projects>

Step 2: Create a project



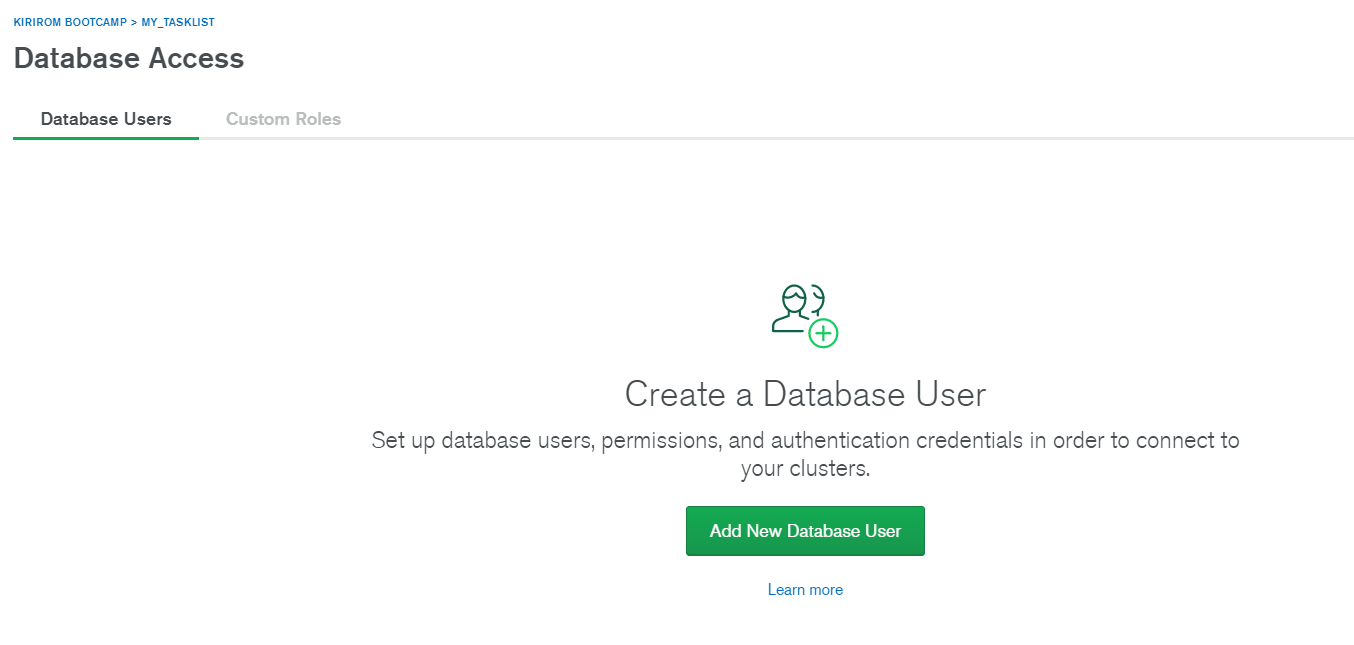
Step 3: Create a cluster

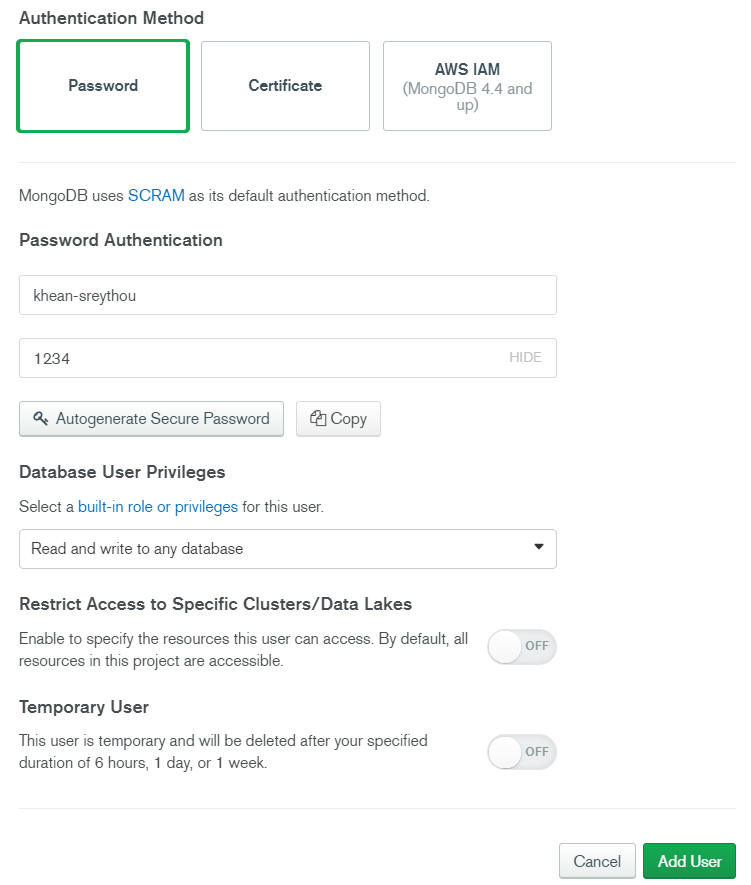
****

Step 4: Add a user

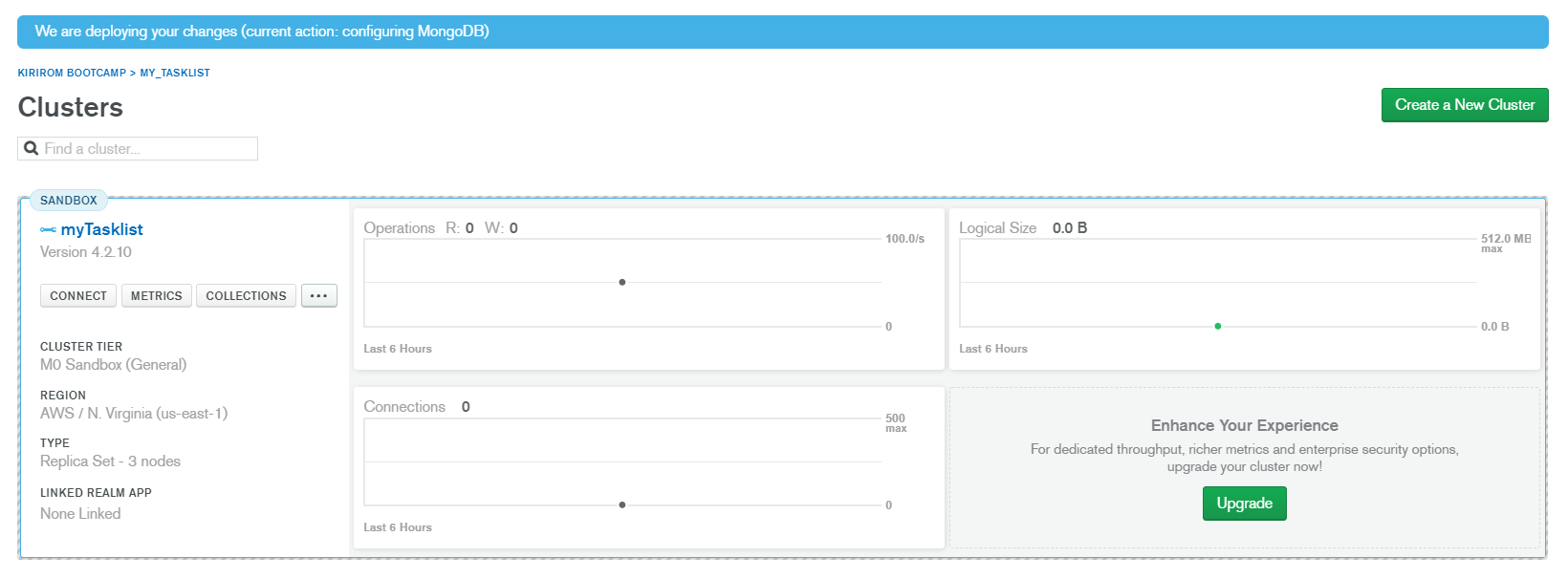
Username : khean-sreythou

Password: 1234

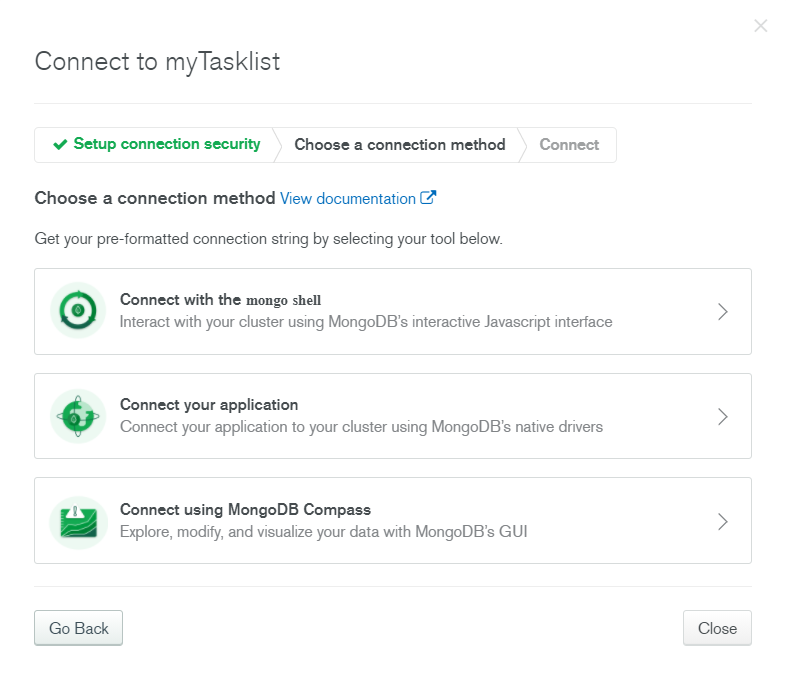




Step 4: Connect cluster: by click ‘CONNECT’ button



Step 5: Choose connection method >> Choose ‘Connect your application’

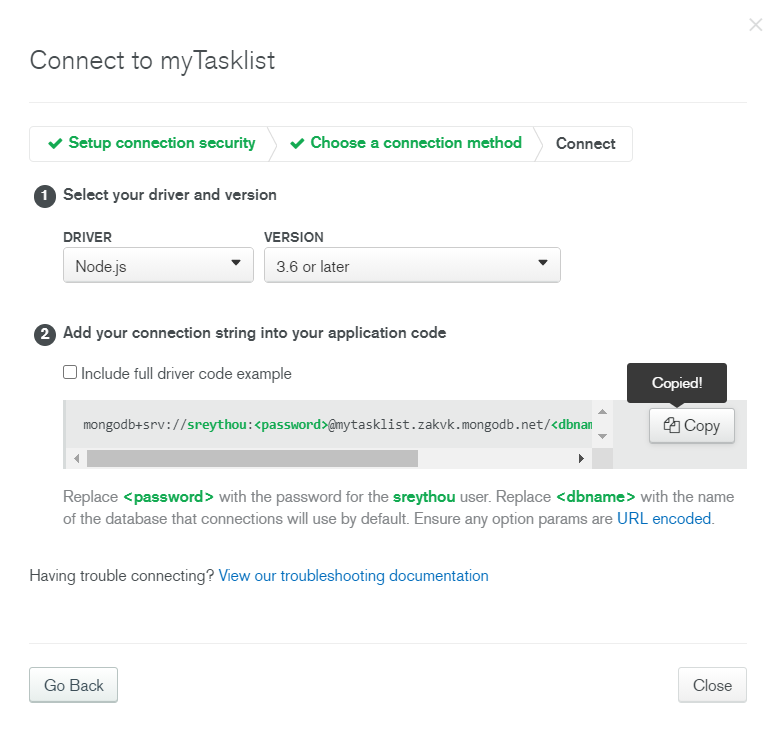
****

Step 6: Choose ‘Driver’ >> Node.js

Choose ‘Version’ >> 3.6 or later

Copy Connection String:

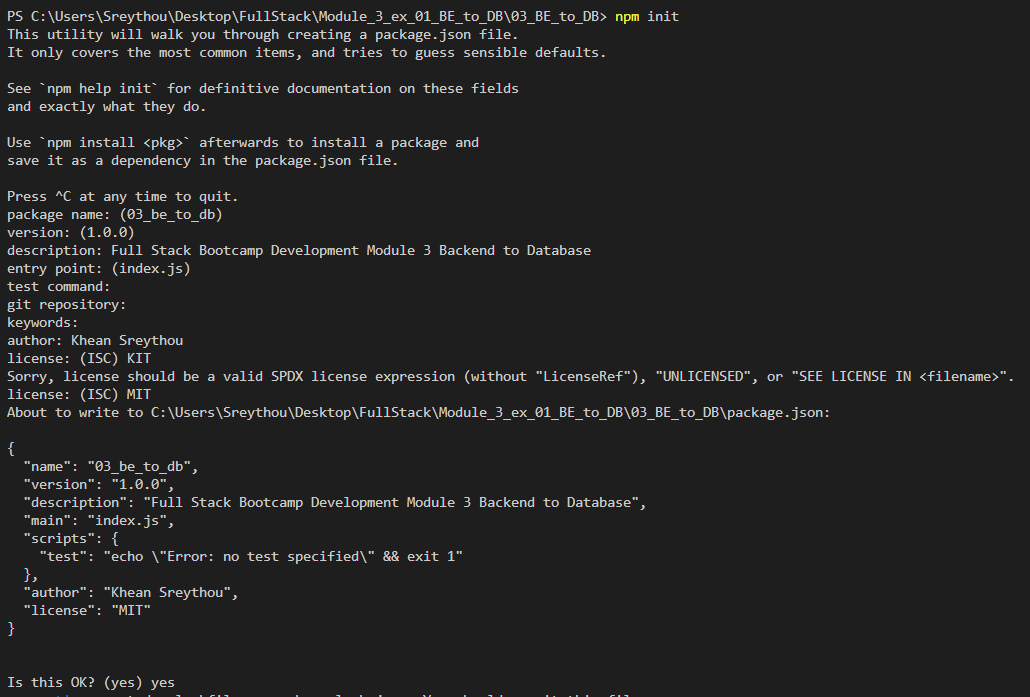
mongodb+srv://sreythou:1234@mytasklist.zakvk.mongodb.net/mytasklist?retryWrites=true&w=majority

****

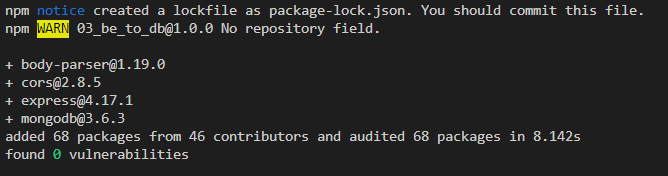
**Setting up the project:**

Step 1: Run *npm init*

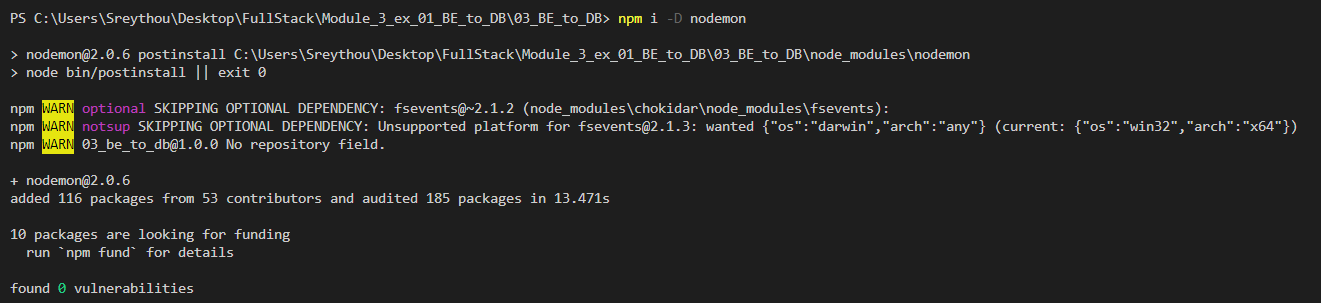
Run it to create a package.json and input some descriptions.



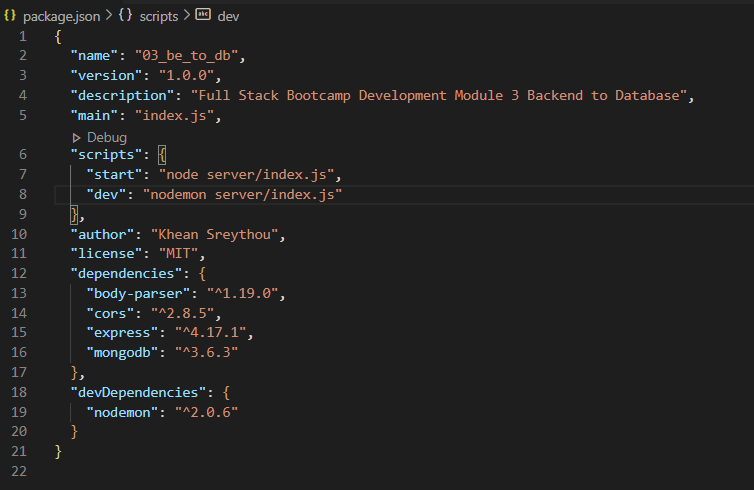
Step 2: Add dependencies: *npm i body-parser cors express mongodb*



Step 3: Add nodemon dependency: *npm i –D nodemon*



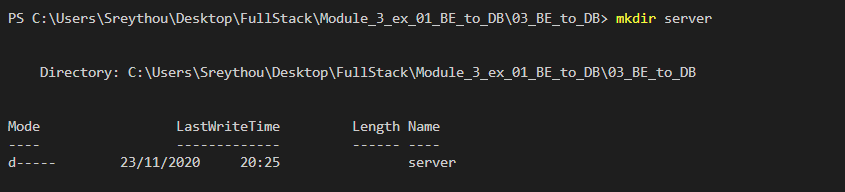
Step 4: In scripts: create start script for running node server and dev script for running –nodemon server



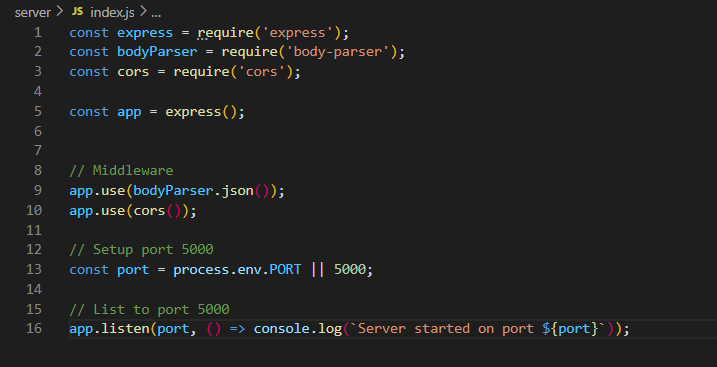
**Setting up the middleware:**

* Create index.js under server folder (server/index.js).
* Setup index.js as the default file for node and nodemon in the package.json file.
* Set up and listen to port 5000 in the index.js file.

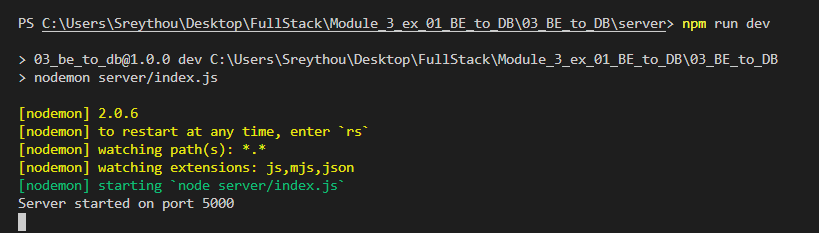
Step 1: Create server folder



Step 2: Inside server folder, create index.js. Then write some code inside it (like the screenshot below).



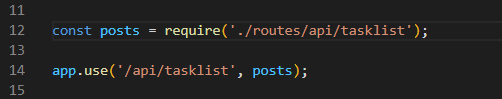
Step 3: Run on server port 5000



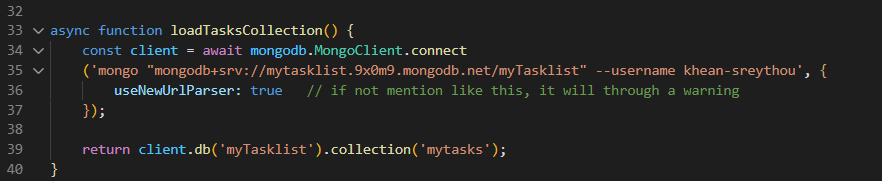
**Creating the API methods:**

Step 1: Inside, **server** folder, create **routes/api/tasklist.js**

Step 2: Create route the request for the URL: api/taskslist to tasklist.js from the middleware

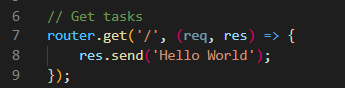


Step 3: Create an async loadTasksCollection() that connect with the database using String URL

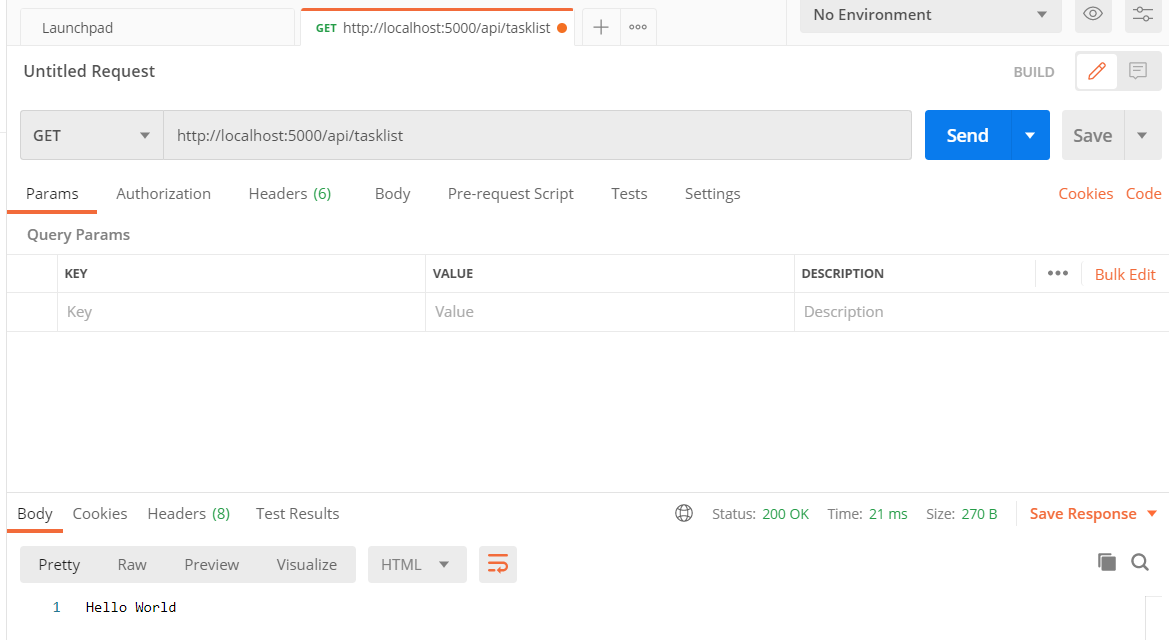


Step 4: Use Postman to test with http request

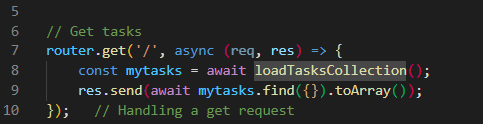
Let’s test by writing this code (below), and use Postman to test either we get that message or not



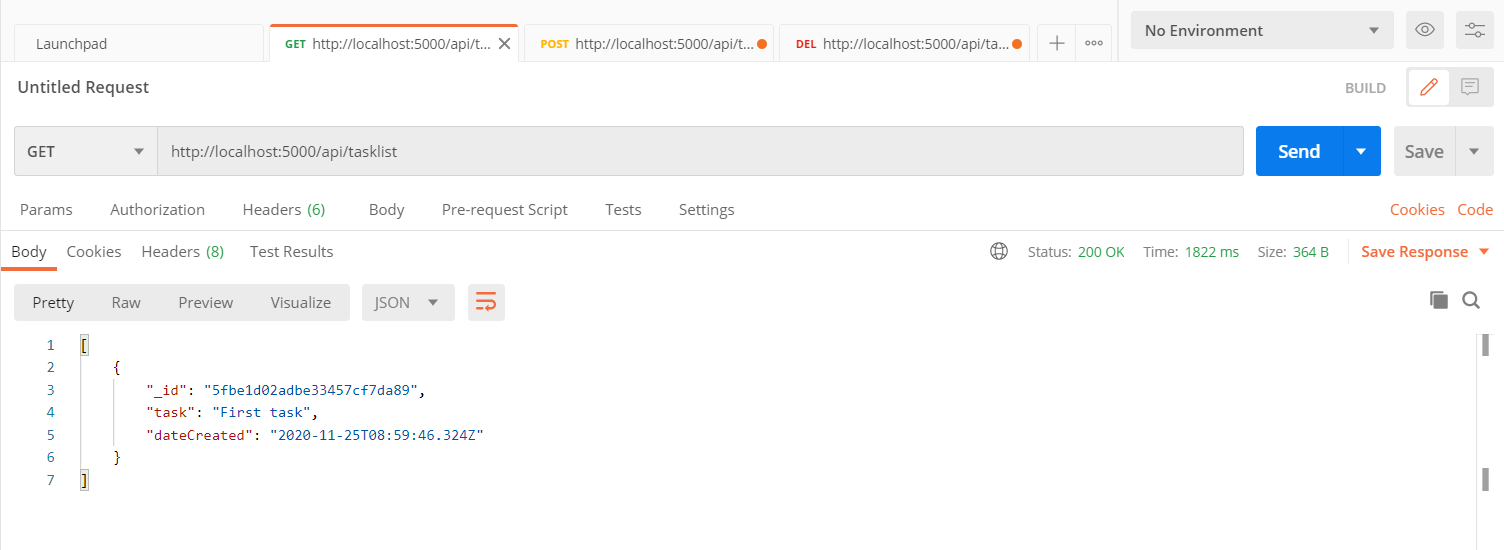
Result in Postman:



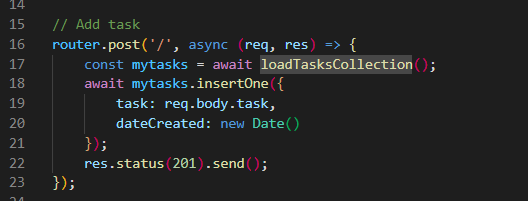
Step 5: Create the router.get() that reads the mytasks collection and return all the tasks.



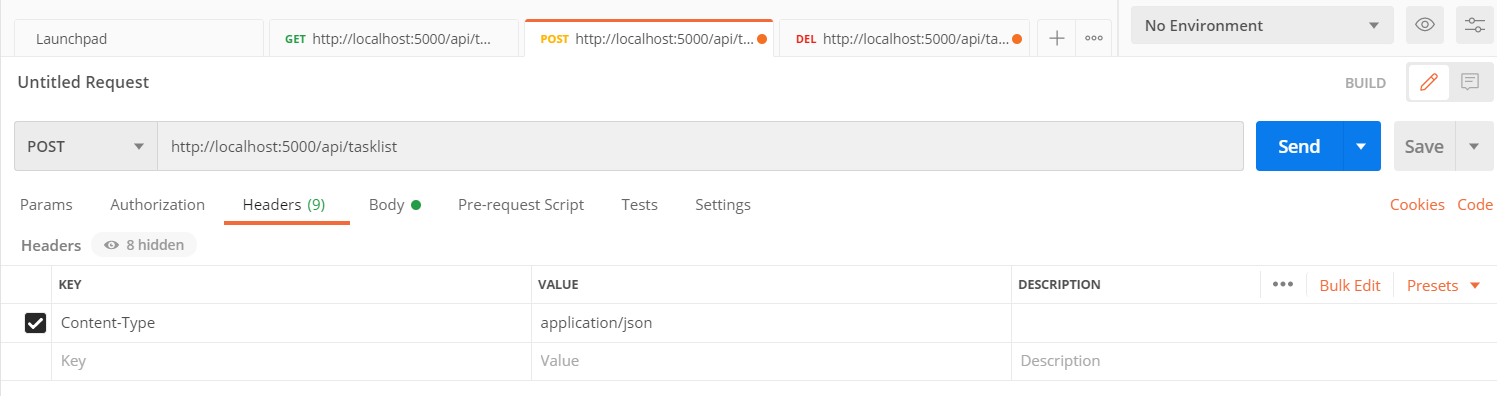
In Postman: Used GET method



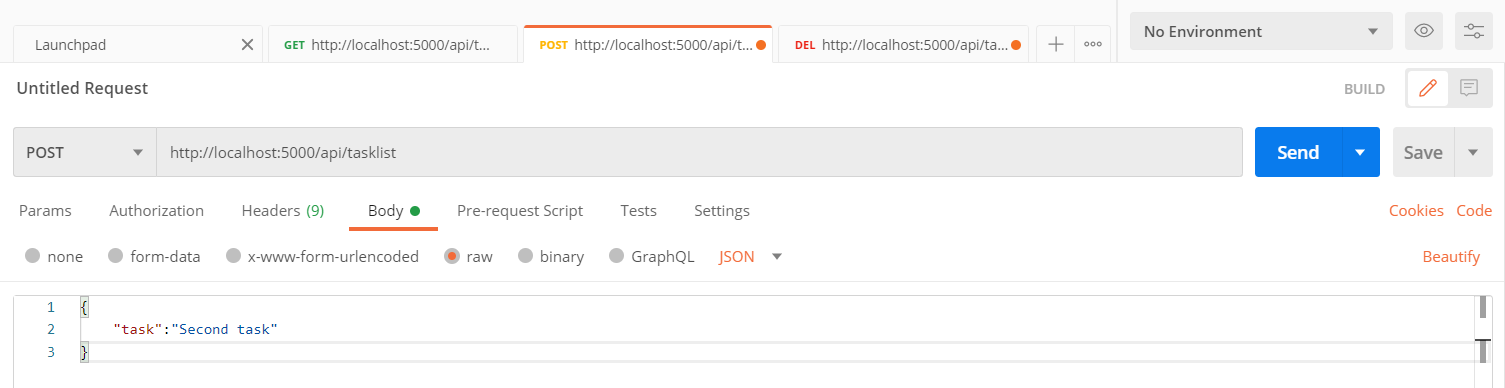
Step 6: Create the router.post() that inserts a new task to the mytasks collection and return the status code 201 if successfully inserted (Fields to be inserted: task and dateCreated). (Routed for the url ‘/’ ).



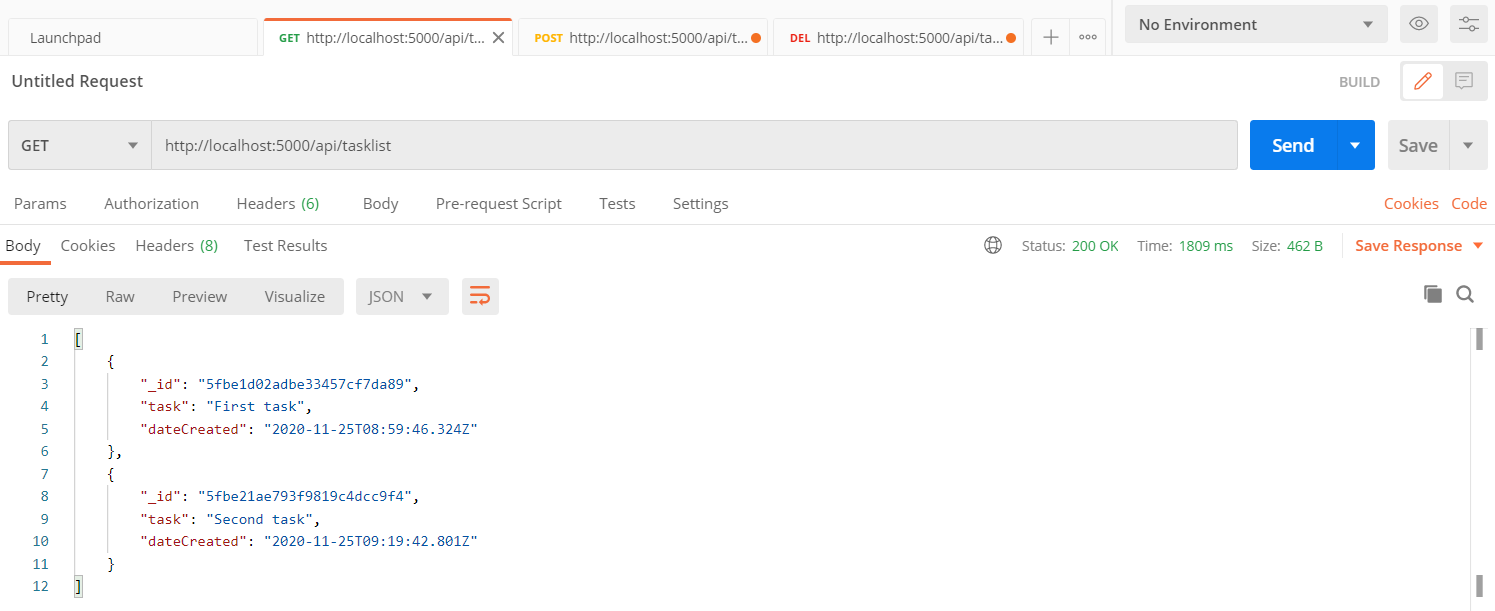
In Postman: First, choose ‘POST’ method. Then create header.



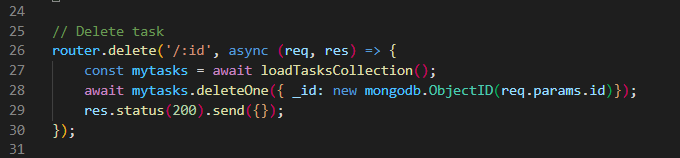
Lastly, create document in Body tab by using raw beautify.



See the result in ‘GET’ method

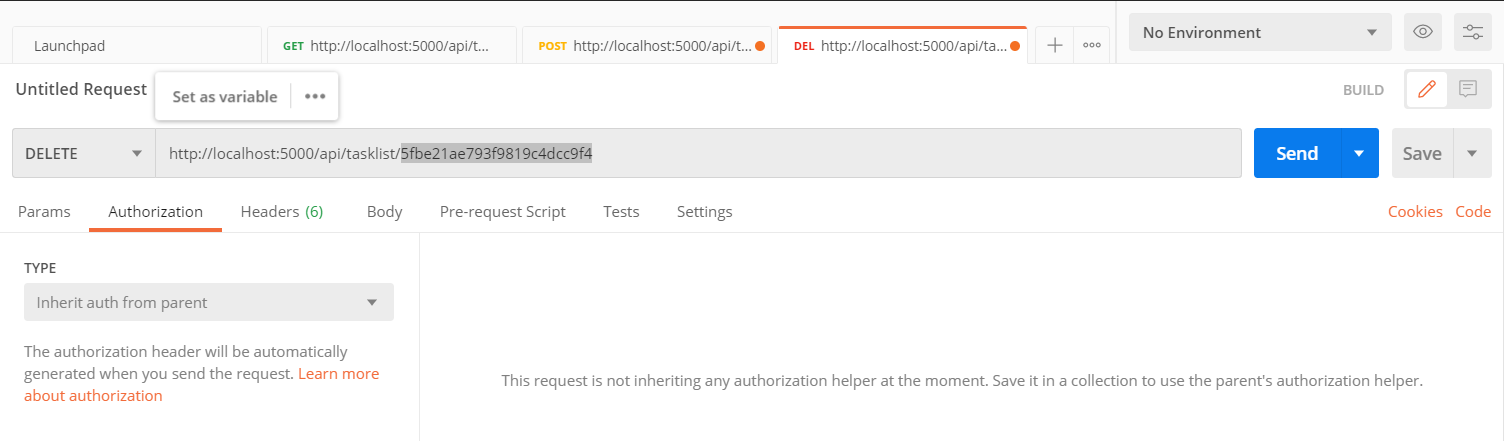


Step 7: Create the router.delete() that deletes an existing task from the mytasks collection and return the status code 200 if successfully deleted. (Routed for the url ‘/<\_id of the task to be deleted>’).

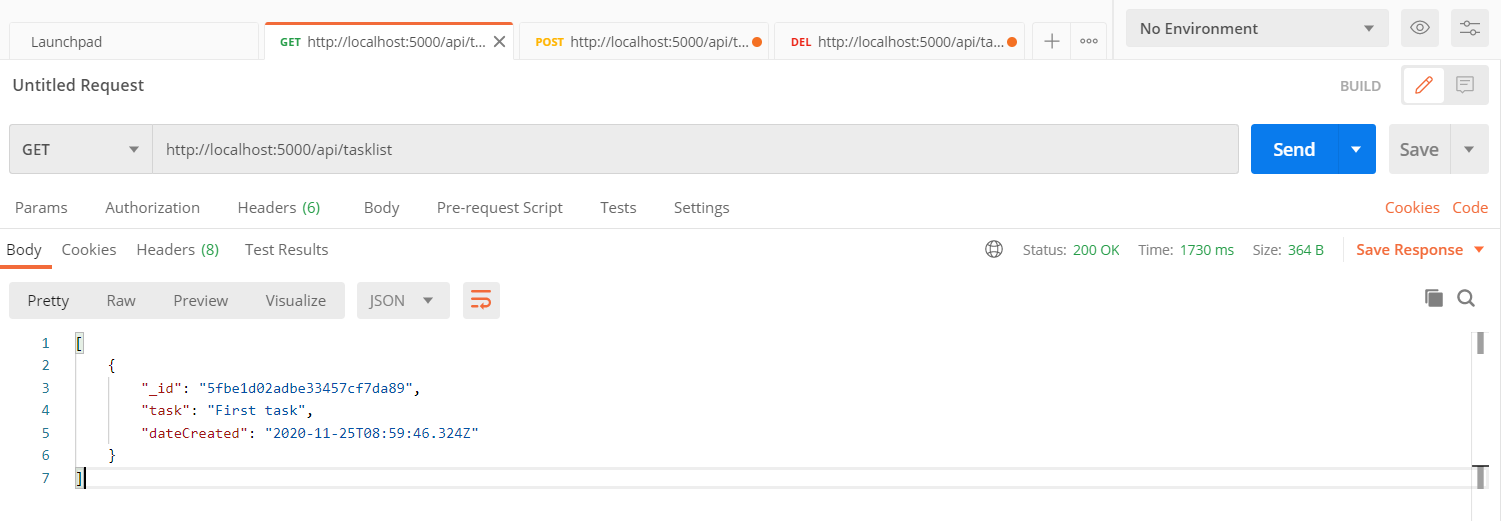


In Postman: Choose ‘DELETE’ method, then pass the id of document along with the ip link.

Example, delete the Second Task document



See the result in ‘GET’ method



**Research Part**

**Different types of dependencies in a Node.js application:**

* Production dependencies
  + Normal dependecies that are mandatory for running the project.
  + These are the packages that are being used in the code of the project.
* Development dependencies
  + Dependencies which are needed at the time of development but are not responsible for working of the application.
  + **Nodemon** is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected.
* Optional dependencies
  + Those dependencies which as the name suggest are optional.
* Peer dependencies
  + These are used when develop a plugin/package for a host tool or package.
  + That means you expect the user of the plugin/package to have these dependencies installed while not necessarily using these dependencies in your plugin/package.
* Bundle dependencies
  + These are array of packages which are bundled when publishing our package.

Package.json vs. Package-lock.json

* Package.json
  + Used to have more information about project.
  + Contains information about all kind of major dependencies.
* Package-lock.json
  + Used to have more information about the dependencies.
  + Contains information about version, integrity, etc of all dependencies.

**Middleware**

*Middleware* functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application’s request-response cycle. The next middleware function is commonly denoted by a variable named next.

Middleware functions can perform the following tasks:

* Execute any code.
* Make changes to the request and the response objects.
* End the request-response cycle.
* Call the next middleware function in the stack.

If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function. Otherwise, the request will be left hanging.

**Postman**

* A software tool to send http requests.
* Useful during real-time development.
* a scalable API testing tool that quickly integrates into CI/CD pipeline.

Why Postman:

1. Accessibility - To use Postman tool, one would just need to log-in to their own accounts making it easy to access files anytime, anywhere as long as a Postman application is installed on the computer.
2. Use of Collections - Postman lets users create collections for their Postman API calls. Each collection can create subfolders and multiple requests. This helps in organizing your test suites.
3. Collaboration - Collections and environments can be imported or exported making it easy to share files. A direct link can also be used to share collections.
4. Creating Environments - Having multiple environments aids in less repetition of tests as one can use the same collection but for a different environment. This is where parameterization will take place which we will discuss in further lessons.
5. Creation of Tests - Test checkpoints such as verifying for successful HTTP response status can be added to each Postman API calls which help ensure [test coverage](https://www.guru99.com/test-coverage-in-software-testing.html).
6. Automation Testing - Through the use of the Collection Runner or Newman, tests can be run in multiple iterations saving time for repetitive tests.
7. Debugging - Postman console helps to check what data has been retrieved making it easy to debug tests.
8. Continuous Integration - With its ability to support continuous integration, development practices are maintained.