PROJECT 3-MERN STACK IMPLEMENTATION

OS used is Ubuntu 20.04 linux distro.

Step 0: Update and upgrade ubuntu

1. `sudo apt update`
2. `sudo apt upgrade`

Step 1: Downloading Node.js Software from ubuntu repo.

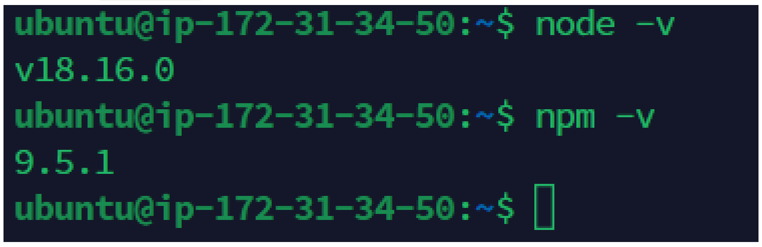
1. `curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -`

Step 2: Install Node.js

1. `sudo apt-get install -y nodejs`
   NB: This command will install Node.js and npm package manager for Node jus as apt is for ubuntu.

Step 3: Verify the Node installation:

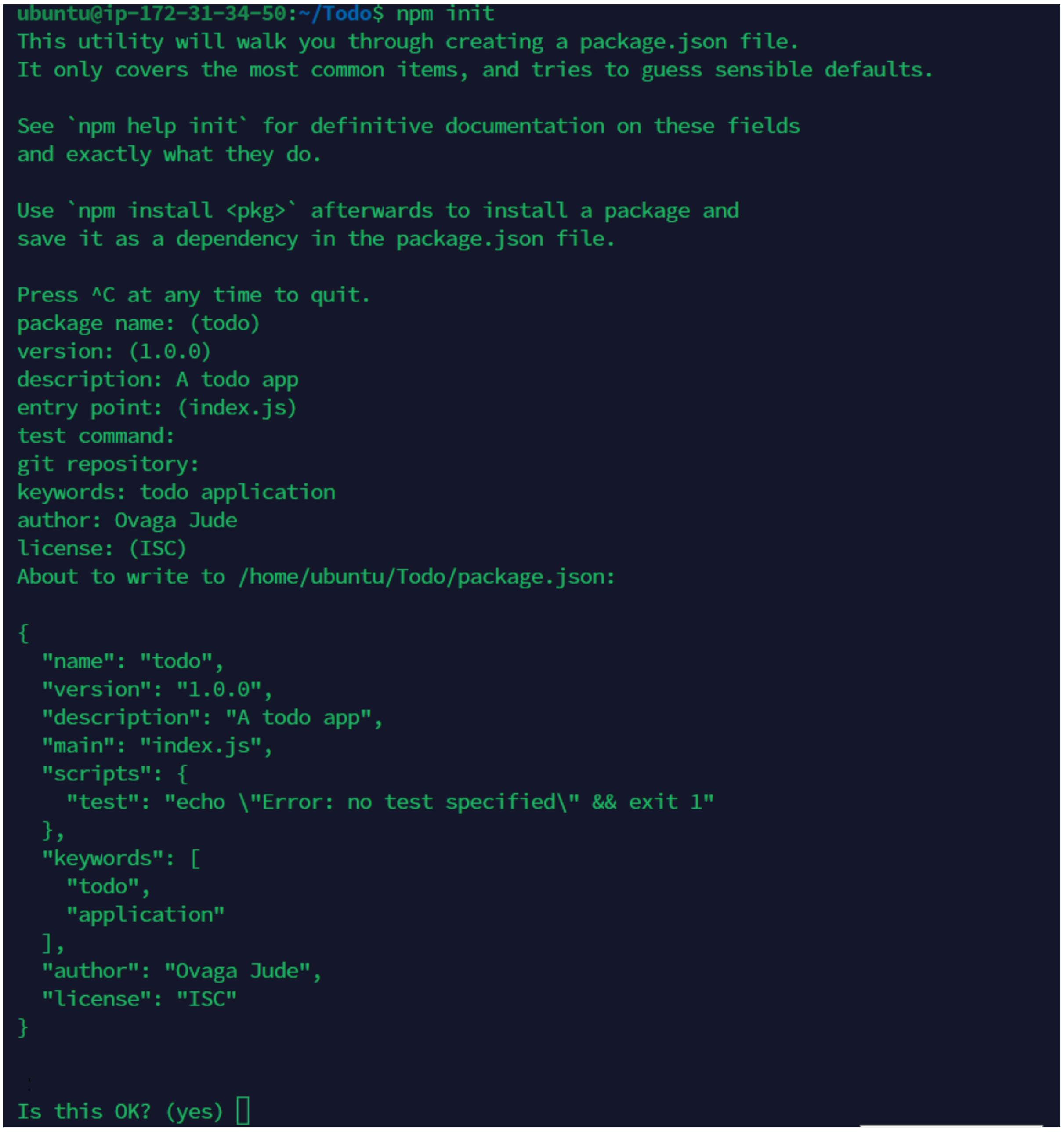
1. `node -v`
2. `npm -v`



```
ubuntu@ip-172-31-34-50:~$ node -v
v18.16.0
ubuntu@ip-172-31-34-50:~$ npm -v
9.5.1
ubuntu@ip-172-31-34-50:~$
```

Step4: Application code set up

1. `Mkdir Todo`
2. `Cd Todo`
3. `Npm init`



```
ubuntu@ip-172-31-34-50:~/Todo$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (todo)
version: (1.0.0)
description: A todo app
entry point: (index.js)
test command:
git repository:
keywords: todo application
author: Ovaga Jude
license: (ISC)
About to write to /home/ubuntu/Todo/package.json:

{
  "name": "todo",
  "version": "1.0.0",
  "description": "A todo app",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "todo",
    "application"
  ],
  "author": "Ovaga Jude",
  "license": "ISC"
}


Is this OK? (yes)
```

Step5: Install expressjs

1. npm install express

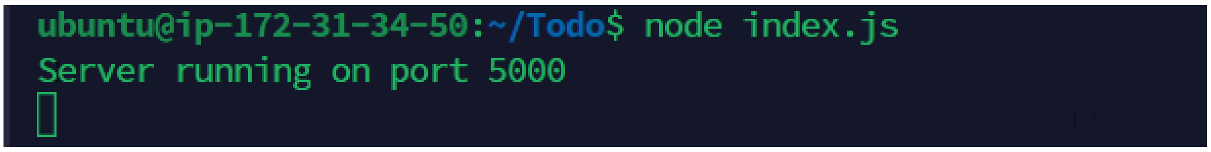
   Creat a file

   1. touch index.js
   2. npm install dotenv
   3. vim index.js

Copy this below code into the index.js file opened and save:

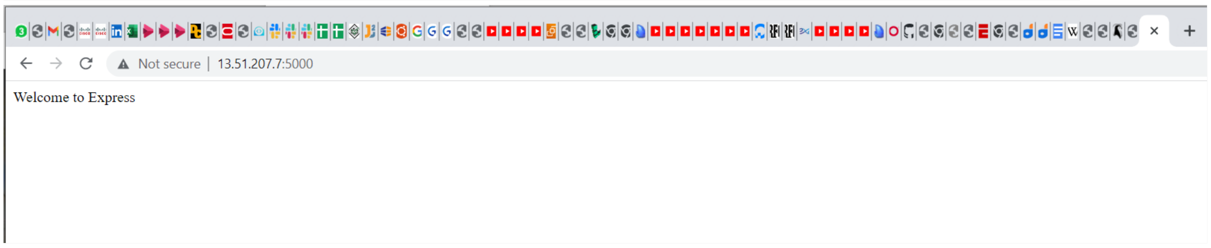
```
const express = require('express');
require('dotenv').config();

const app = express();

const port = process.env.PORT || 5000;
```

```
app.use((req, res, next) => {
res.header("Access-Control-Allow-Origin", "\*");
res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With,
Content-Type, Accept");
next();
});

app.use((req, res, next) => {
res.send('Welcome to Express');
});

app.listen(port, () => {
console.log(`Server running on port ${port}`)
});

Step6: Start the server to test if everything is working well:
Run : node index.js
```



```
ubuntu@ip-172-31-34-50:~/Todo$ node index.js
Server running on port 5000
```

```
Open the port 5000 in the Server and confirm it is working fine:
```



Welcome to Express

Routes

There are three actions that our To-Do application needs to be able to do:

1. Create a new task
2. Display list of all tasks
3. Delete a completed task

Each task will be associated with some particular endpoint and will use different standard HTTP request methods: POST, GET, DELETE.

For each task, we need to create `routes` that will define various endpoints that the

`To-do` app will depend on. So let us create a folder `routes`

```
Step7:

Mkdir routes

Cd routes

Touch api.js

Vim api.js

const express = require ('express');

const router = express.Router();

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

});

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

});

router.delete('/todos/:id', (req, res, next) => {

})

module.exports = router;
```

```
CREATING MODELS
```

Now comes the interesting part, since the app is going to make use of Mongodb which is a NoSQL database, we need to create a model.

A model is at the heart of JavaScript based applications, and it is what makes it interactive.

We will also use models to define the database schema . This is important so that we will be able to define the fields stored in each Mongodb document

In essence, the Schema is a blueprint of how the database will be constructed, including other data fields that may not be required to be stored in the database. These are known as *virtual properties*

To create a Schema and a model, install *mongoose* which is a Node.js package that makes working with mongodb easier.

`Step8:` Change directory back Todo folder with `cd ..`and install Mongoose

1. Cd ..
2. `npm install mongoose`
3. `mkdir models`
4. `cd models`
5. Vi todo.js

```
const mongoose = require('mongoose');

const Schema = mongoose.Schema;

//create schema for todo

const TodoSchema = new Schema({

action: {

type: String,

required: [true, 'The todo text field is required']

}

})

//create model for todo

const Todo = mongoose.model('todo', TodoSchema);
```

```
module.exports = Todo;
```

Now we need to update our routes from the file `api.js` in 'routes' directory to make use of the new model.In Routes directory, open api.js with `vim api.js`, delete the code inside with `:%d` command and paste there code below into it then save and exit

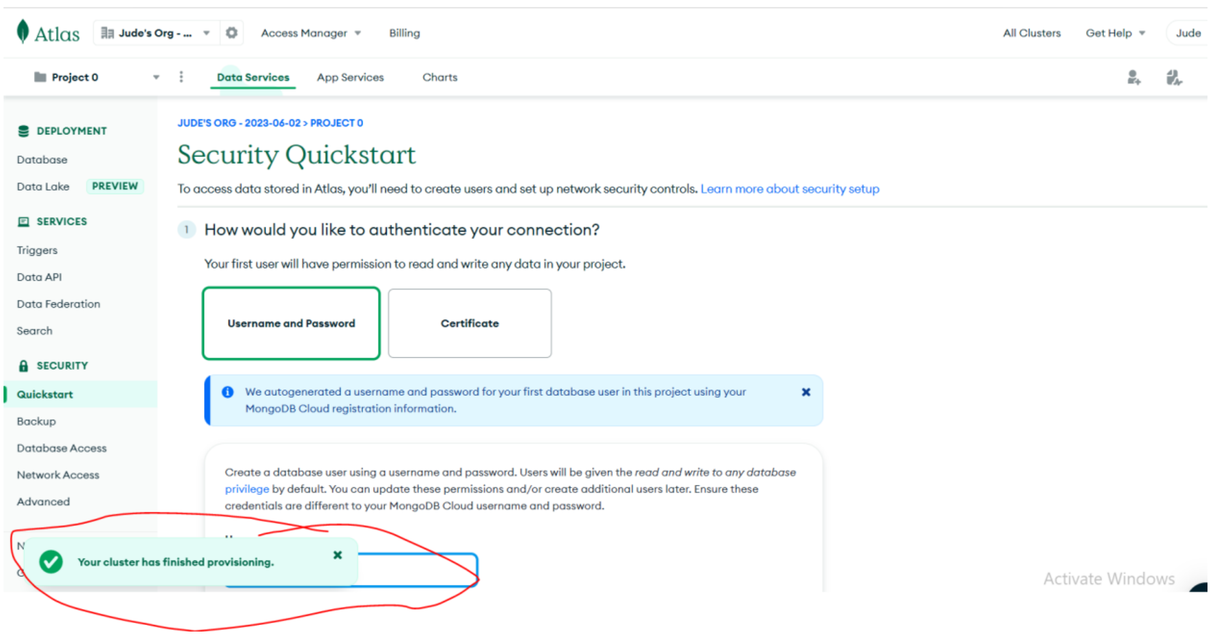
```
const express = require ('express');

const router = express.Router();

const Todo = require('../models/todo');
```

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

//this will return all the data, exposing only the id and action field
to the client

Todo.find({}, 'action')

.then(data => res.json(data))

.catch(next)

});

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

if(req.body.action){

Todo.create(req.body)

.then(data => res.json(data))

.catch(next)

}else {

res.json({

error: "The input field is empty"

})

}

});

router.delete('/todos/:id', (req, res, next) => {

Todo.findOneAndDelete({"_id": req.params.id})

.then(data => res.json(data))

.catch(next)

})

module.exports = router;
```

MONGODB DATABASE

MongoDB Database
We need a database where we will store our data. For this we will make use of **mLab**. mLab provides MongoDB database as a service solution (DBaaS), so to make life easy, you will need to sign up for a shared clusters free account, which is ideal for our use case. Sign up here. Follow the sign up process, select **AWS** as the cloud provider, and choose a region near you.

Step1: Creat a Cluster



JUDE'S ORG - 2023-06-02 > PROJECT 0

Security Quickstart

To access data stored in Atlas, you'll need to create users and set up network security controls. Learn more about security setup

1 How would you like to authenticate your connection?

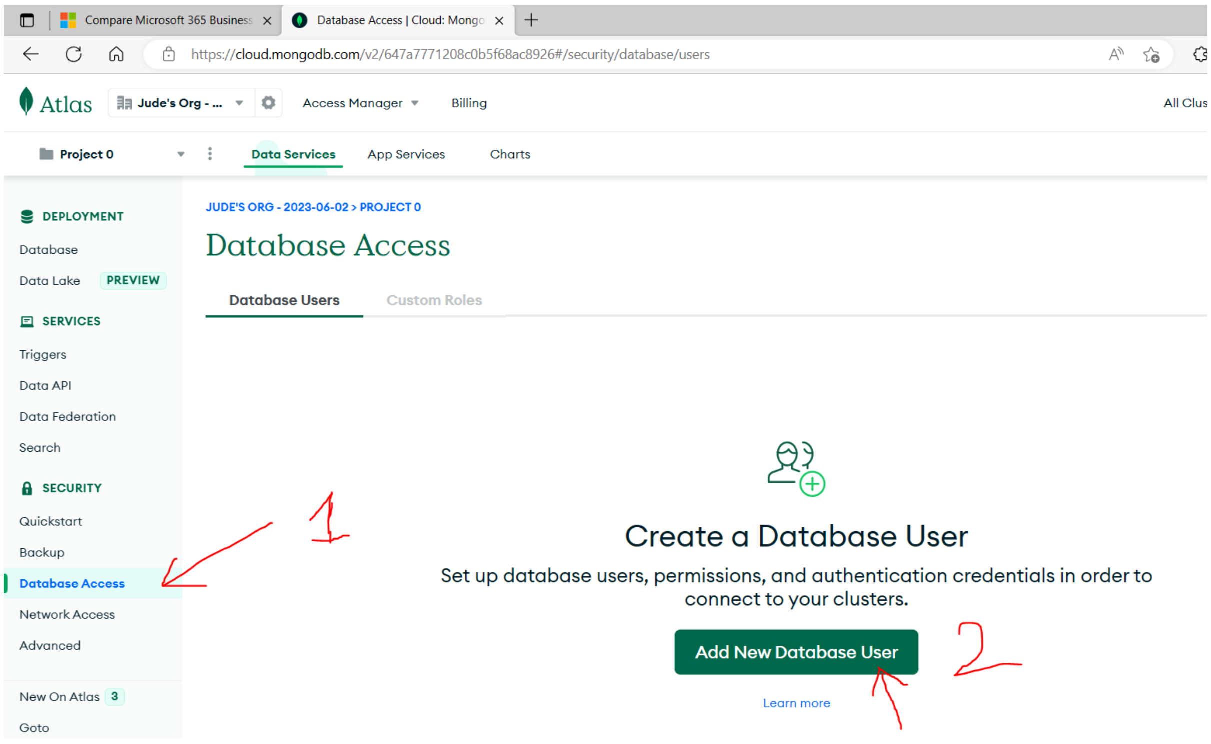
Your first user will have permission to read and write any data in your project.

We autogenerated a username and password for your first database user in this project using your MongoDB Cloud registration information.

Create a database user using a username and password. Users will be given the *read and write to any database* privilege by default. You can update these permissions and/or create additional users later. Ensure these credentials are different to your MongoDB Cloud username and password.

Your cluster has finished provisioning.

Step2: Add a new Database USER


https://cloud.mongodb.com/v2/647a7771208c0b5f68ac8926#/security/database/users

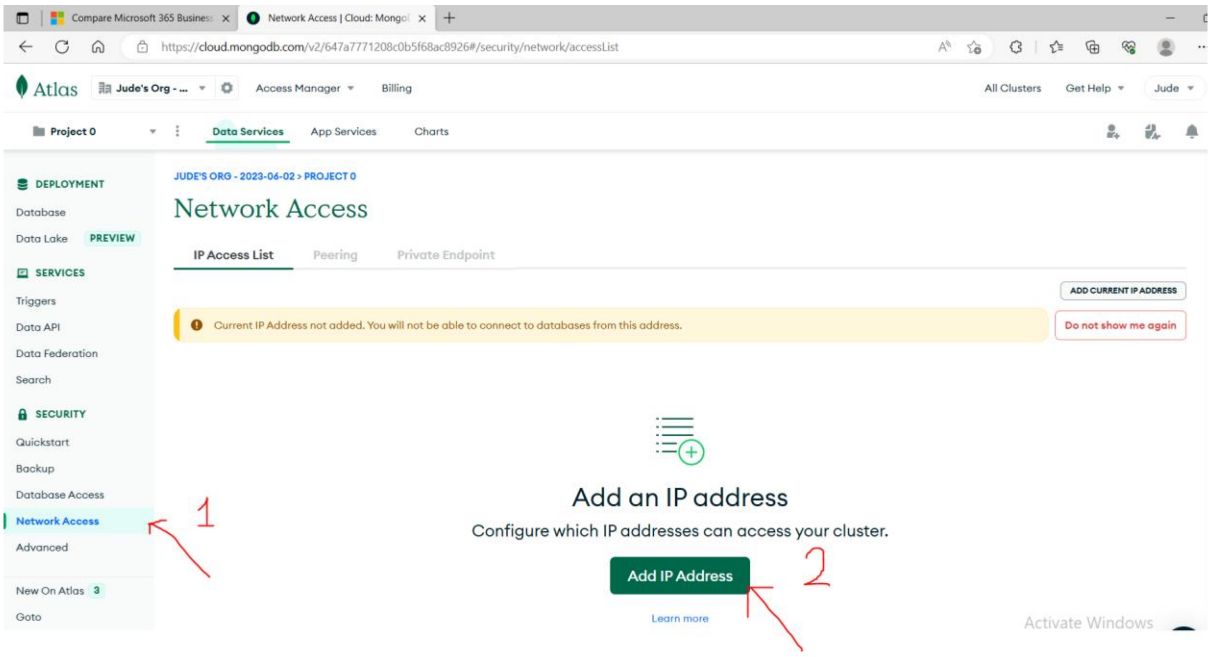
JUDE'S ORG - 2023-06-02 > PROJECT 0

Database Access

Create a Database User

Set up database users, permissions, and authentication credentials in order to connect to your clusters.

Network Access Configuration


https://cloud.mongodb.com/v2/647a7771208c0b5f68ac8926#/security/network/accessList

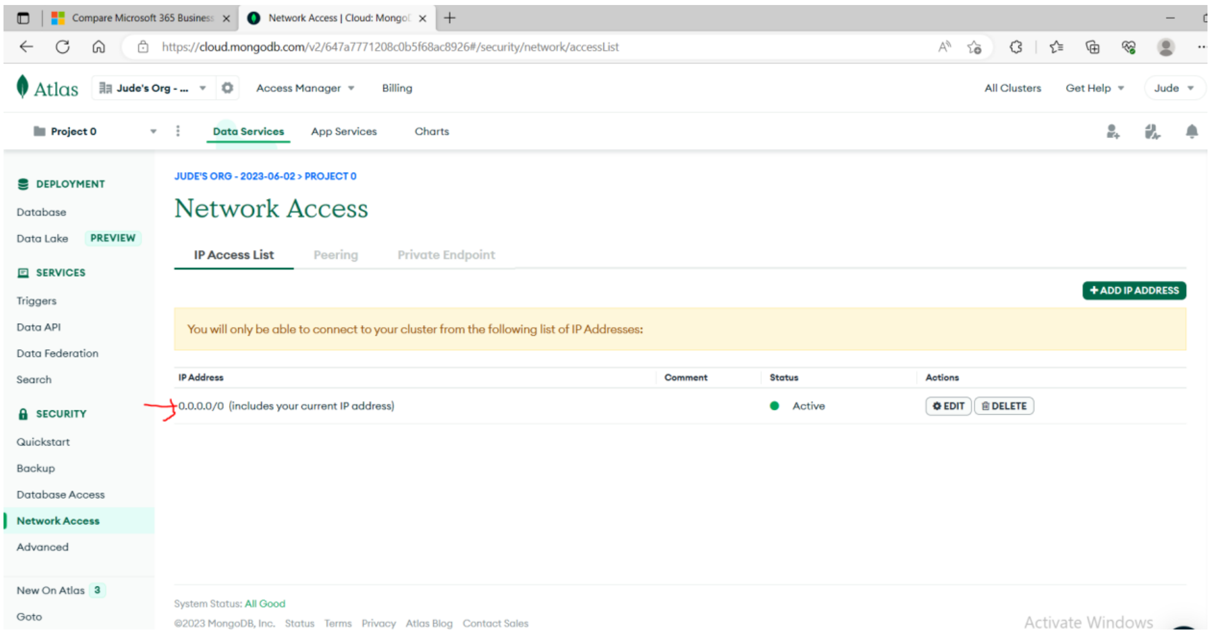
JUDE'S ORG - 2023-06-02 > PROJECT 0

Network Access

Current IP Address not added. You will not be able to connect to databases from this address.

Add an IP address

Configure which IP addresses can access your cluster.



JUDE'S ORG - 2023-06-02 > PROJECT 0

Network Access

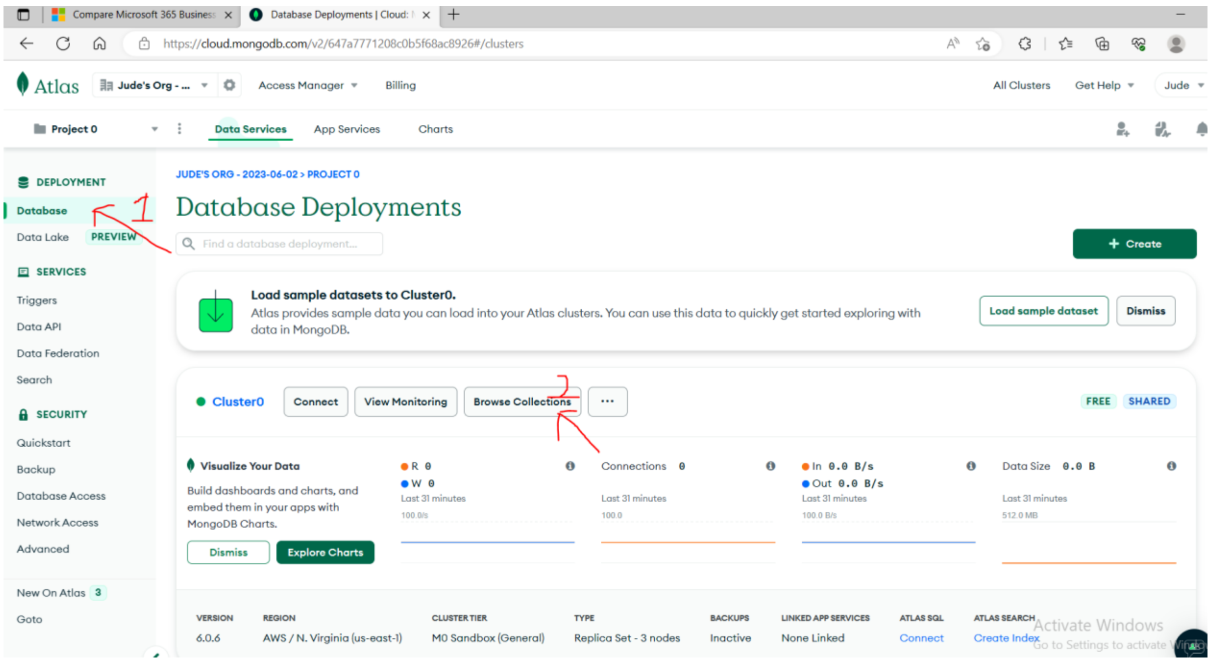
You will only be able to connect to your cluster from the following list of IP Addresses:

IP Address	Comment	Status	Actions
0.0.0.0/0 (includes your current IP address)		● Active	EDIT DELETE

System Status: All Good

©2023 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales

Step3: Create a MongoDB database and Collections



JUDE'S ORG - 2023-06-02 > PROJECT 0

Database Deployments

Load sample datasets to Cluster0.
Atlas provides sample data you can load into your Atlas clusters. You can use this data to quickly get started exploring with data in MongoDB.

Visualize Your Data
Build dashboards and charts, and embed them in your apps with MongoDB Charts.

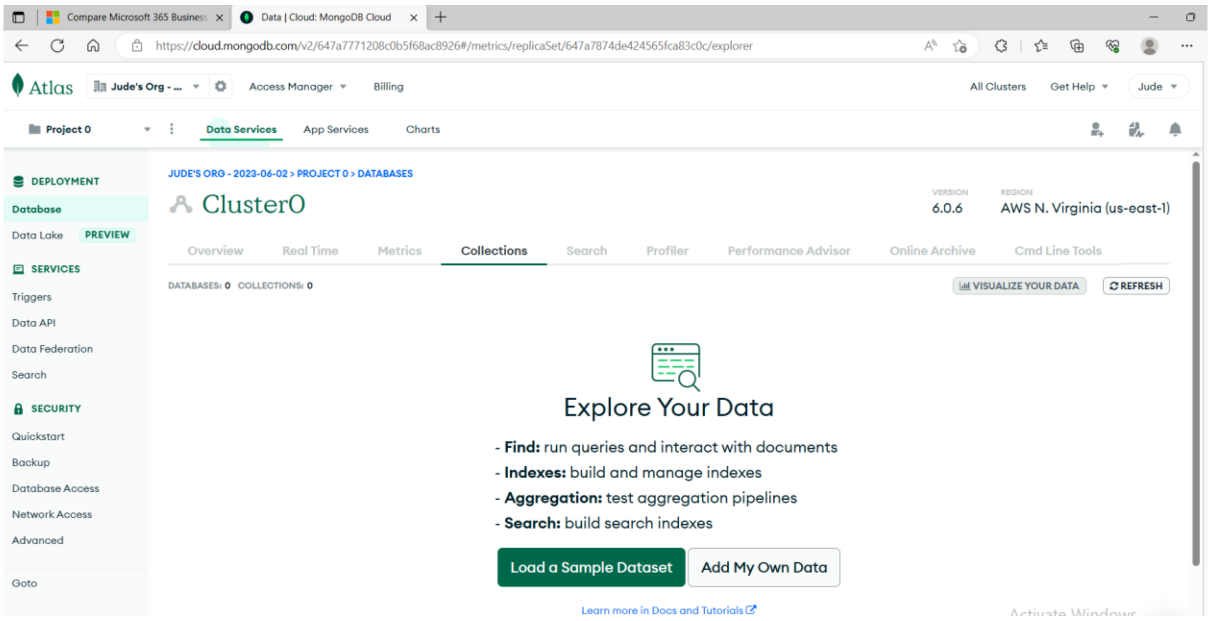
R 0
W 0
Last 31 minutes
100.0/s

Connections 0
Last 31 minutes
100.0

In 0.0 B/s
Out 0.0 B/s
Last 31 minutes
100.0 B/s

Data Size 0.0 B
Last 31 minutes
512.0 MB

VERSION	REGION	CLUSTER TIER	TYPE	BACKUPS	LINKED APP SERVICES	ATLAS SQL	ATLAS SEARCH
6.0.6	AWS / N. Virginia (us-east-1)	M0 Sandbox (General)	Replica Set - 3 nodes	Inactive	None Linked	Connect	Create Index


Compare Microsoft 365 Business | Data | Cloud: MongoDB Cloud

https://cloud.mongodb.com/v2/647a7771208c0b5f68ac8926#/metrics/replicaSet/647a7874de424565fca83c0c/explorer

Atlas | Jude's Org - ... | Access Manager | Billing | All Clusters | Get Help | Jude

Project 0 | Data Services | App Services | Charts

DEPLOYMENT
- Database
- Data Lake PREVIEW

SERVICES
- Triggers
- Data API
- Data Federation
- Search

SECURITY
- Quickstart
- Backup
- Database Access
- Network Access
- Advanced

Goto

JUDE'S ORG - 2023-06-02 > PROJECT 0 > DATABASES

ClusterO

VERSION 6.0.6 | REGION AWS N. Virginia (us-east-1)

Overview | Real Time | Metrics | Collections | Search | Profiler | Performance Advisor | Online Archive | Cmd Line Tools

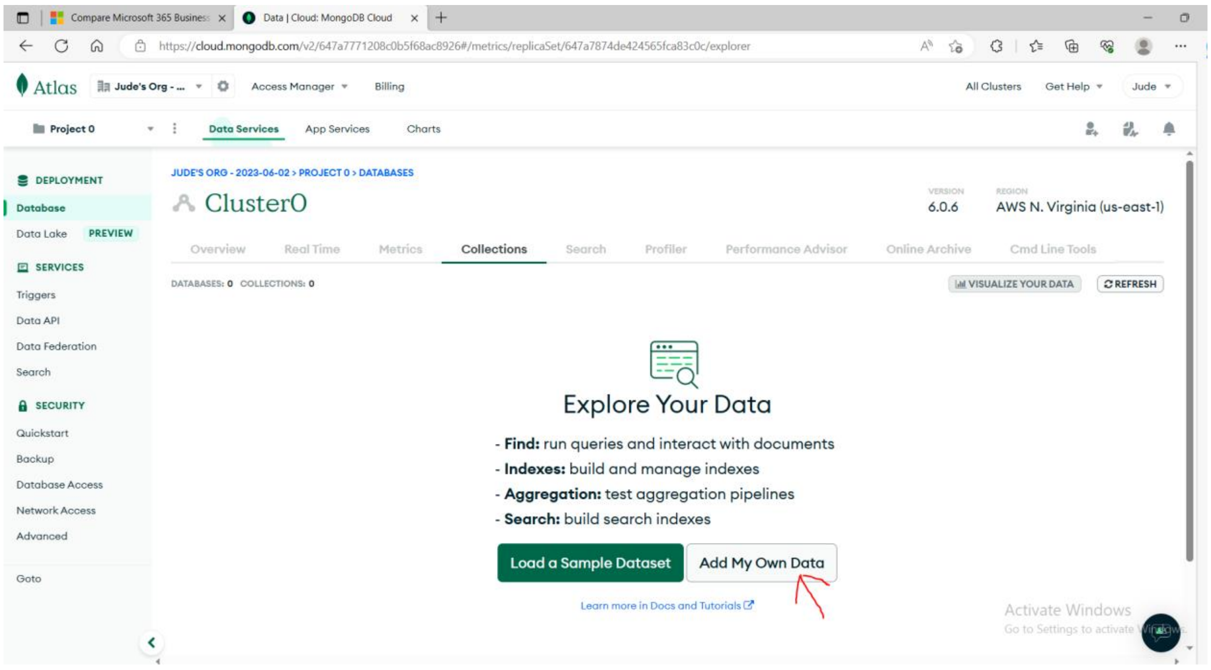
DATABASES: 0 COLLECTIONS: 0 | VISUALIZE YOUR DATA | REFRESH

Explore Your Data

- **Find:** run queries and interact with documents
- **Indexes:** build and manage indexes
- **Aggregation:** test aggregation pipelines
- **Search:** build search indexes

Load a Sample Dataset | Add My Own Data

Learn more in Docs and Tutorials


Compare Microsoft 365 Business | Data | Cloud: MongoDB Cloud

https://cloud.mongodb.com/v2/647a7771208c0b5f68ac8926#/metrics/replicaSet/647a7874de424565fca83c0c/explorer

Atlas | Jude's Org - ... | Access Manager | Billing | All Clusters | Get Help | Jude

Project 0 | Data Services | App Services | Charts

DEPLOYMENT
- Database
- Data Lake PREVIEW

SERVICES
- Triggers
- Data API
- Data Federation
- Search

SECURITY
- Quickstart
- Backup
- Database Access
- Network Access
- Advanced

Goto

JUDE'S ORG - 2023-06-02 > PROJECT 0 > DATABASES

ClusterO

VERSION 6.0.6 | REGION AWS N. Virginia (us-east-1)

Overview | Real Time | Metrics | Collections | Search | Profiler | Performance Advisor | Online Archive | Cmd Line Tools

DATABASES: 0 COLLECTIONS: 0 | VISUALIZE YOUR DATA | REFRESH

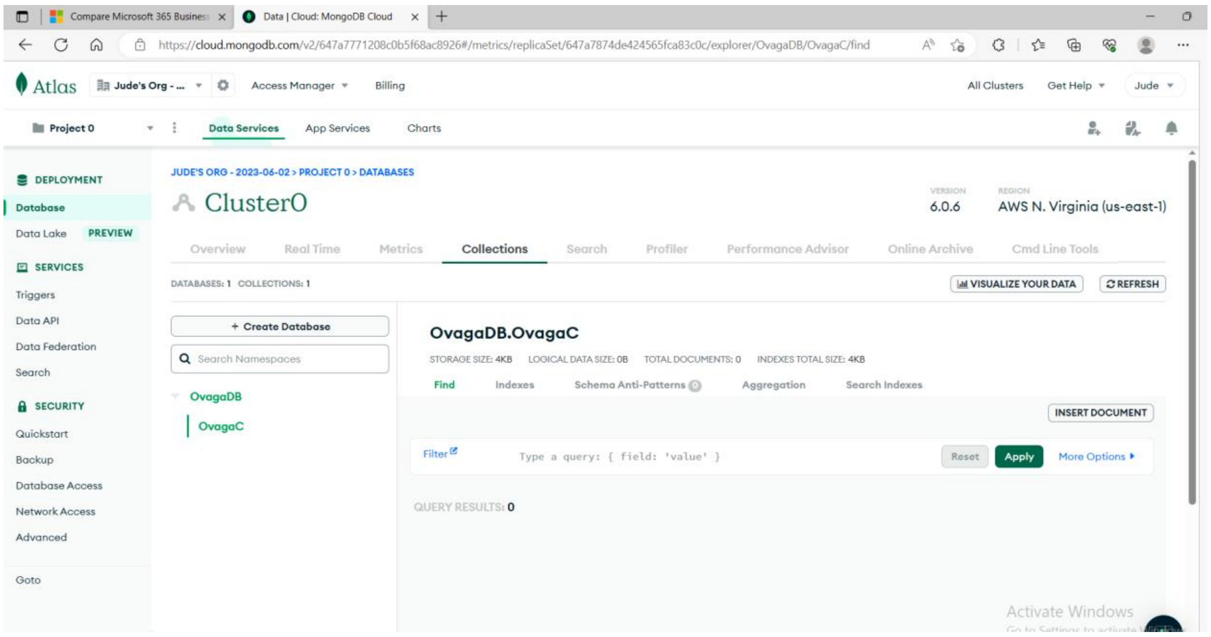
Explore Your Data

- **Find:** run queries and interact with documents
- **Indexes:** build and manage indexes
- **Aggregation:** test aggregation pipelines
- **Search:** build search indexes

Load a Sample Dataset | Add My Own Data

Learn more in Docs and Tutorials

Activate Windows
Go to Settings to activate Windows.



In the `index.js` file, we specified `process.env` to access environment variables, but we have not yet created this file. So we need to do that now.Create a file in your `Todo` directory and name it `.env`

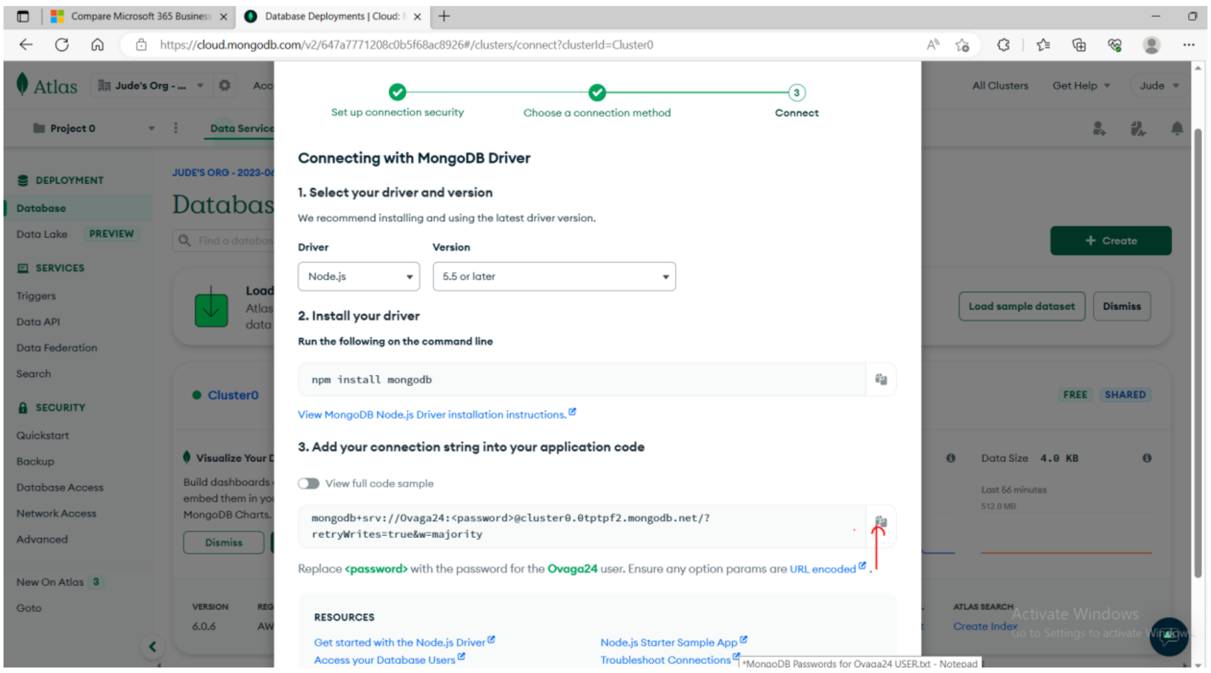
```
Step1:

Cd Todo

Sudo vim .env
```

Add the connection string to access the database in it, just as below:

```
DB = 'mongodb+srv://<username>:<password>@<network-
address>/<dbname>?retryWrites=true&w=majority'
```

Ensure to update `<username><password><network-address>` and `<database>` according to your setup

To get connection string for the Database go to below:



Set up connection security — Choose a connection method — Connect

Connecting with MongoDB Driver

1. Select your driver and version

We recommend installing and using the latest driver version.

Driver: Node.js
Version: 5.5 or later

2. Install your driver

Run the following on the command line

```
npm install mongodb
```

View MongoDB Node.js Driver installation instructions.

3. Add your connection string into your application code

View full code sample

```
mongodb+srv://Ovaga24:<password>@cluster0.0tptpf2.mongodb.net/?
retryWrites=true&w=majority
```

Replace **<password>** with the password for the **Ovaga24** user. Ensure any option params are URL encoded .

RESOURCES

Get started with the Node.js Driver
Node.js Starter Sample App
Access your Database Users
Troubleshoot Connections

Now we need to update the `index.js` to reflect the use of `.env` so that Node.js can connect to the database.Simply delete existing content in the file, and update it with the entire code below.

```
const express = require('express');

const bodyParser = require('body-parser');

const mongoose = require('mongoose');

const routes = require('./routes/api');

const path = require('path');

require('dotenv').config();



const app = express();



const port = process.env.PORT || 5000;



//connect to the database
```

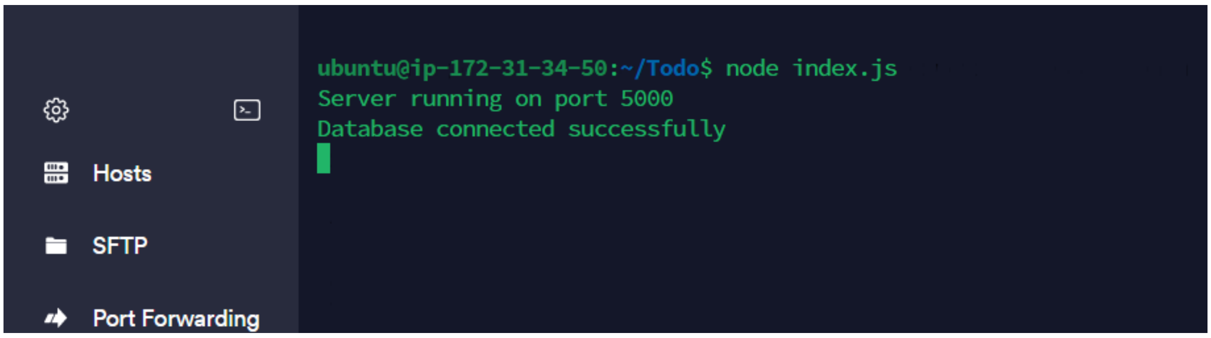
```
});



mongoose.connect(process.env.DB, { useNewUrlParser: true,
useUnifiedTopology: true })

.then(() => console.log(`Database connected successfully`))

.catch(err => console.log(err));



//since mongoose promise is depreciated, we overide it with node's
promise

mongoose.Promise = global.Promise;



app.use((req, res, next) => {

res.header("Access-Control-Allow-Origin", "\*");

res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With,
Content-Type, Accept");

next();

});



app.use(bodyParser.json());



app.use('/api', routes);



app.use((err, req, res, next) => {

console.log(err);

next();

});



app.listen(port, () => {
```

```
console.log(`Server running on port ${port}`)

});
```

NB: Using environment variables to store information is considered more secure and best practice to separate configuration and secret data from the application, instead of writing connection strings directly inside the `index.js` application file.

Step1: Start the node.js server

```
node index.js
```



```
ubuntu@ip-172-31-34-50:~/Todo$ node index.js
Server running on port 5000
Database connected successfully
```

So the backend configurations are good. We will need to test it.

NB: So far we have written the backend part of our `To-Do` application, and configured a database, but we do not have a frontend UI yet. We need ReactJS code to achieve that. But during development, we will need a way to test our code using RESTfull API. Therefore, we will need to make use of some API development client to test our code.

In this project, we will use Postman to test our API.

Step1: Download Postman

NB: Learn how perform CRUD operations on Postman.

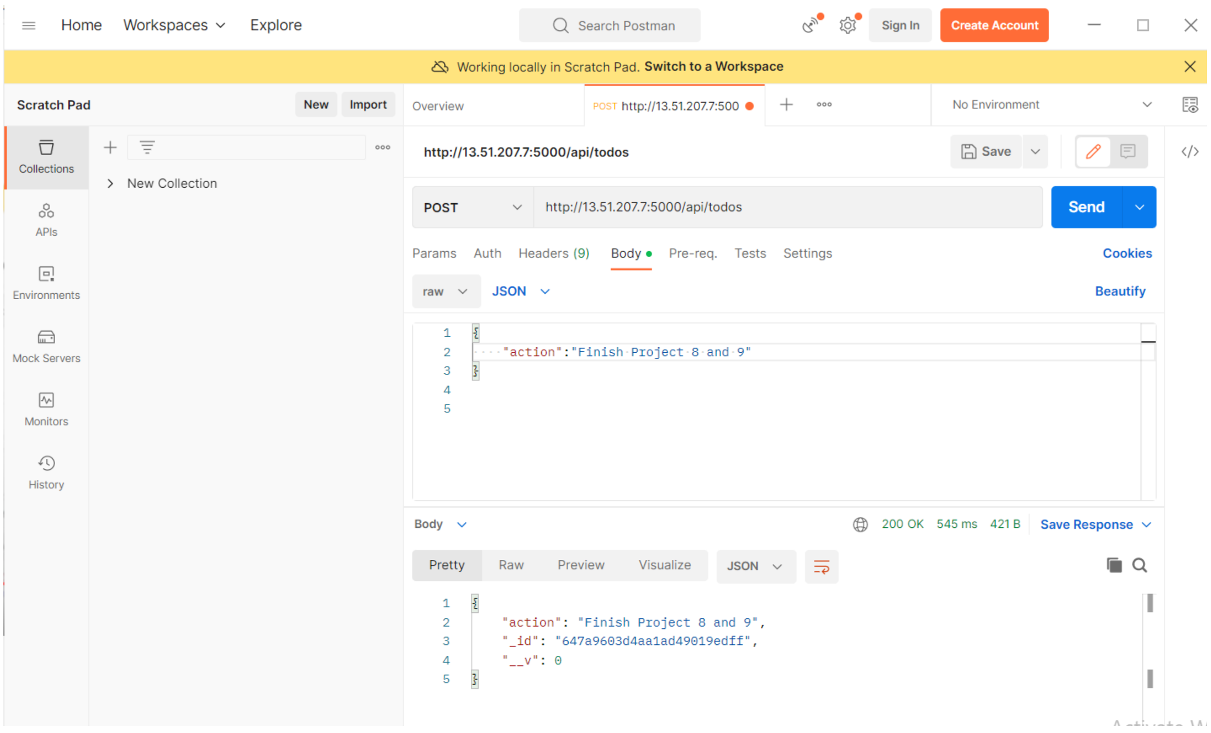
You should test all the API endpoints and make sure they are working. For the endpoints that require body, you should send JSON back with the necessary fields since it's what we setup in our code.

Now open your Postman, create a POST request to the API

`http://<PublicIP-or-PublicDNS>:5000/api/todos`

. This request sends a new task to our To-Do list so the application could store it in the database.

Note: make sure your set header key `Content-Type` as `application/json`



http://13.51.207.7:5000/api/todos

POST http://13.51.207.7:5000/api/todos

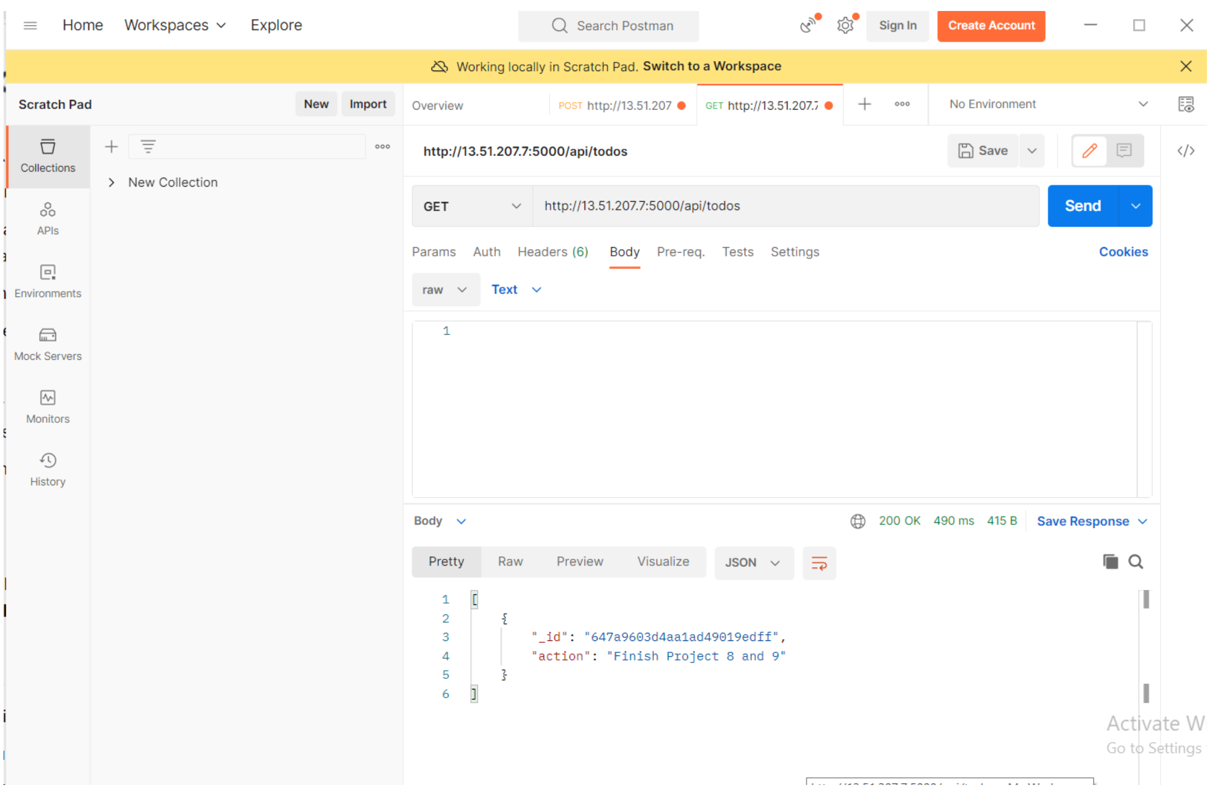
```
{
    "action":"Finish Project 8 and 9"
}
```

Body 200 OK 545 ms 421 B

```
{
    "action": "Finish Project 8 and 9",
    "_id": "647a9603d4aa1ad49019edff",
    "__v": 0
}
```

Create a GET request to your API on

`http://<PublicIP-or-PublicDNS>:5000/api/todos`. This request retrieves all existing records from out To-do application (backend requests these records from the database and sends it us back as a response to GET request)



http://13.51.207.7:5000/api/todos

GET http://13.51.207.7:5000/api/todos

Body 200 OK 490 ms 415 B

```
[
    {
        "_id": "647a9603d4aa1ad49019edff",
        "action": "Finish Project 8 and 9"
    }
]
```

So the backend is working fine as expected.

Phase 2: Frontend Creation

Since we are done with the functionality we want from our backend and API, it is time to create a user interface for a Web client (browser) to interact with the application via API. To start out with the frontend of the To-do app, we will use the `Create-react-app` command to scaffold our app.

In the same root directory as your backend code, which is the Todo directory, run:

```
npx create-react-app client
```

This will create a new folder in your `Todo` directory called `client`, where you will add all the react code.

Running a React App

Before testing the react app, there are some dependencies that need to be installed.

1. Install concurrently. It is used to run more than one command simultaneously from the same terminal window.

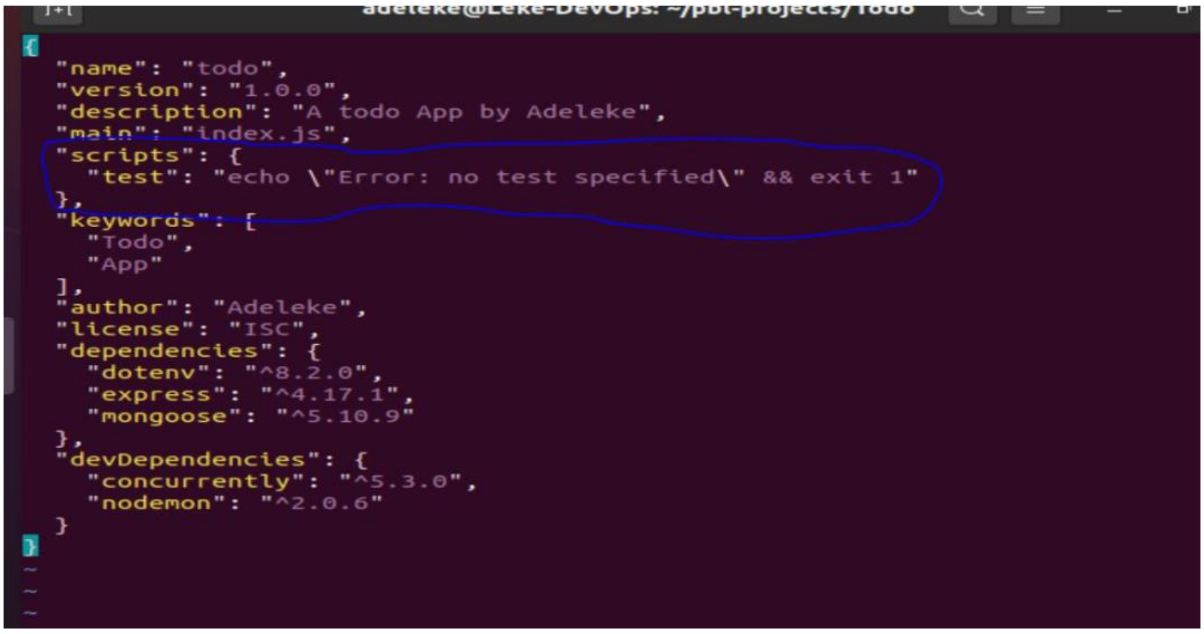
```
npm install concurrently --save-dev
```

2. Install nodemon. It is used to run and monitor the server. If there is any change in the server code, nodemon will restart it automatically and load the new changes.

```
      npm install nodemon --save-dev
```

In `Todo` folder open the `package.json` file. Change the highlighted part of the below screenshot and replace with the code below.

```
"scripts": {"start": "node index.js","start-watch": "nodemon
index.js","dev": "concurrently \"npm run start-watch\" \"cd client &&
npm start\""

},
```



```
{
  "name": "todo",
  "version": "1.0.0",
  "description": "A todo App by Adeleke",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "Todo",
    "App"
  ],
  "author": "Adeleke",
  "license": "ISC",
  "dependencies": {
    "dotenv": "^8.2.0",
    "express": "^4.17.1",
    "mongoose": "^5.10.9"
  },
  "devDependencies": {
    "concurrently": "^5.3.0",
    "nodemon": "^2.0.6"
  }
}
~
~
~
```

Configure Proxy in `package.json`

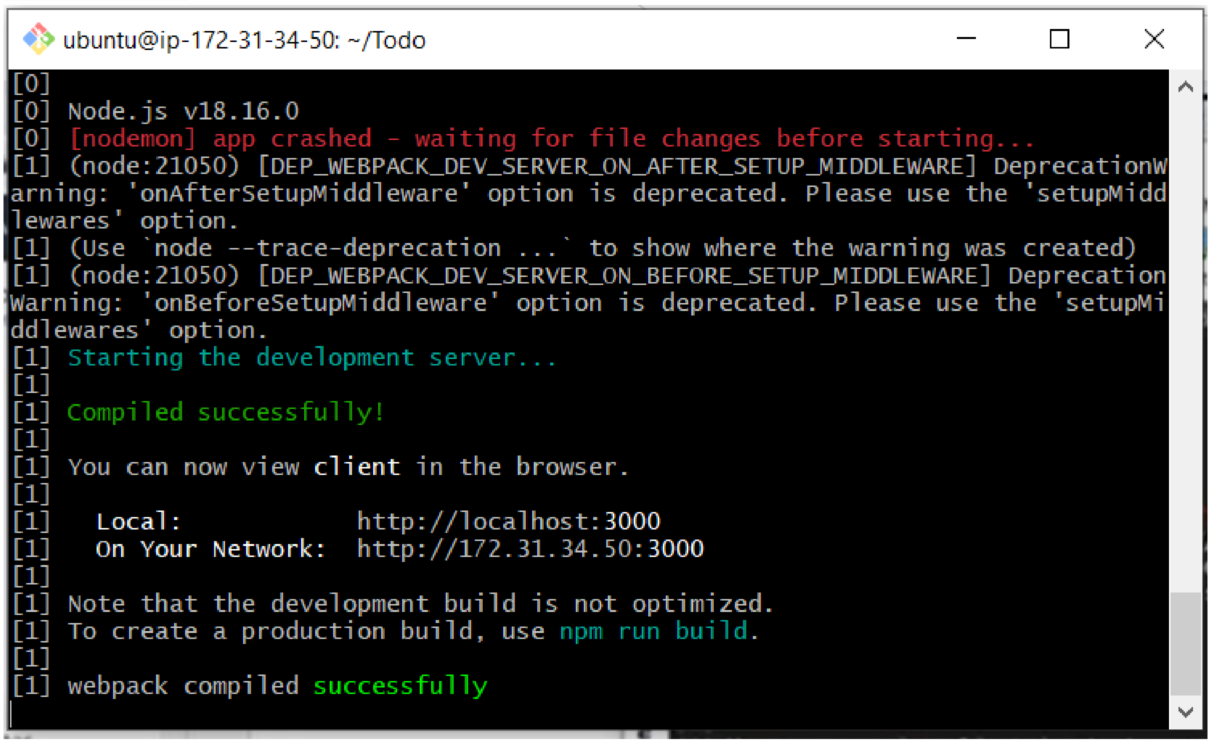
```
cd client
vi package.json
```

Add the key value pair in the package.json file `"proxy": "http://localhost:5000"`

The whole purpose of adding the proxy configuration in number 3 above is to make it possible to access the application directly from the browser by simply calling the server url like `http://localhost:5000` rather than always including the entire path like [`http://localhost:5000/api/todos`](http://localhost:5000/api/todos)

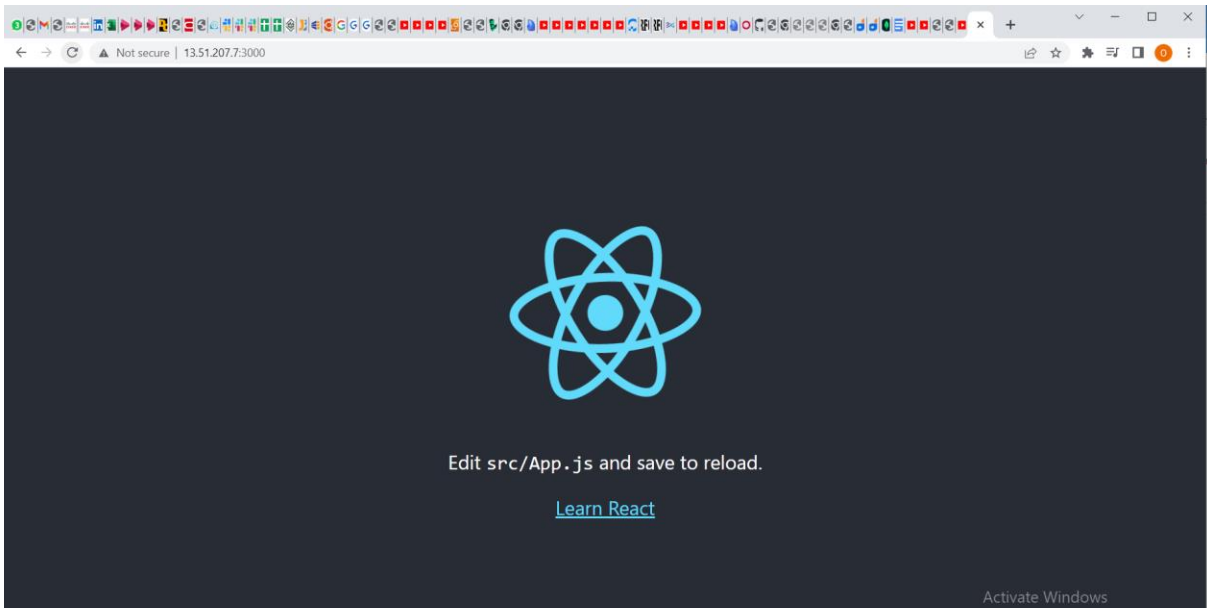
Now, ensure you are inside the `Todo` directory, and simply do:

```
npm run dev
```



```
ubuntu@ip-172-31-34-50: ~/Todo
[0]
[0] Node.js v18.16.0
[0] [nodemon] app crashed - waiting for file changes before starting...
[1] (node:21050) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
[1] (Use `node --trace-deprecation ...` to show where the warning was created)
[1] (node:21050) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
[1] Starting the development server...
[1]
[1] Compiled successfully!
[1]
[1] You can now view client in the browser.
[1]
[1]   Local:            http://localhost:3000
[1]   On Your Network:  http://172.31.34.50:3000
[1]
[1] Note that the development build is not optimized.
[1] To create a production build, use npm run build.
[1]
[1] webpack compiled successfully
```

Your app should open and start running on `localhost:3000`


Not secure | 13.51.207.7:3000
Edit src/App.js and save to reload.
Learn React
Activate Windows

Creating your React Components

One of the advantages of react is that it makes use of components, which are reusable and also makes code modular. For our Todo app, there will be two stateful components and one stateless component.

From your Todo directory run

```
cd client
cd src
mkdir components
cd components
```

Inside ‘components’ directory create three files `Input.js`, `ListTodo.js` and `Todo.js`

```
touch Input.js ListTodo.js Todo.js
vi Input.js
```

```
import React, { Component } from 'react';

import axios from 'axios';



class Input extends Component {



state = {

action: ""

}



addTodo = () => {

const task = {action: this.state.action}



    if(task.action && task.action.length > 0){

      axios.post('/api/todos', task)

        .then(res => {
          if(res.data){

            this.props.getTodos();

            this.setState({action: ""})

          }

        })

        .catch(err => console.log(err))

    }else {

      console.log('input field required')

    }


}


handleChange = (e) => {

this.setState({

action: e.target.value

})

}


render() {

let { action } = this.state;

return (

<div>
```

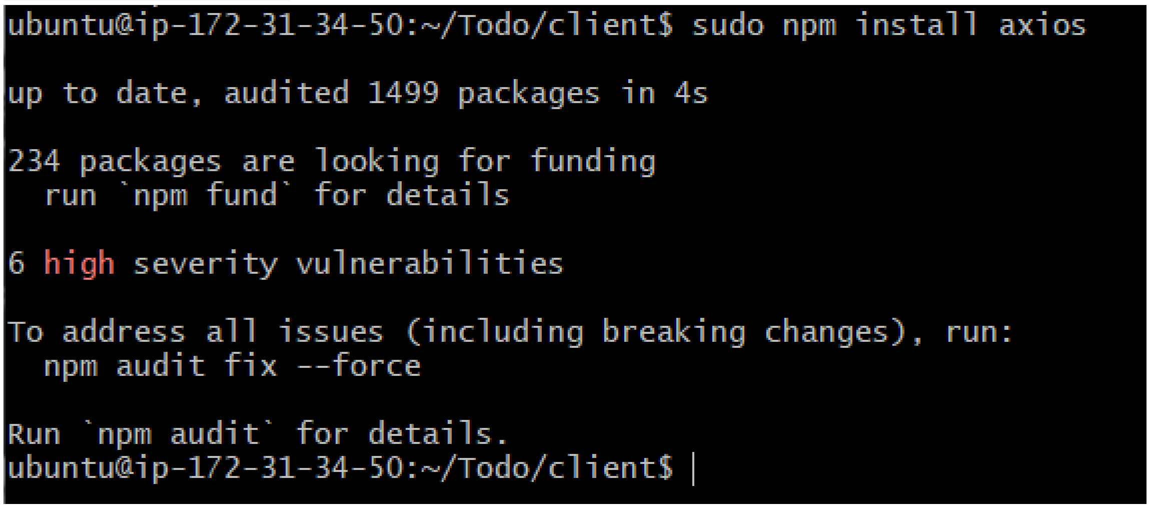
```
<input type="text" onChange={this.handleChange} value={action} />

<button onClick={this.addTodo}>add todo</button>

</div>

)

}

}



export default Input
```

To make use of Axios, which is a Promise based HTTP client for the browser and node.js, you need to cd into your client from your terminal and run yarn add axios or npm install axios.

```
cd ..
cd ..
```

Install Axios

```
npm install axios
```



```
ubuntu@ip-172-31-34-50:~/Todo/client$ sudo npm install axios

up to date, audited 1499 packages in 4s

234 packages are looking for funding
  run `npm fund` for details

6 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
ubuntu@ip-172-31-34-50:~/Todo/client$ |
```

```
cd src/components
```

```
vi ListTodo.js
import React from 'react';



const ListTodo = ({ todos, deleteTodo }) => {



return (

<ul>

{

todos &&

todos.length > 0 ?

(

todos.map(todo => {

return (

<li key={todo._id} onClick={() =>
deleteTodo(todo._id)}>{todo.action}</li>

)

})

)

:

(

<li>No todo(s) left</li>

)

}
</ul>

)

}


export default ListTodo
```

Vim Todo.js

```
import React, {Component} from 'react';

import axios from 'axios';


import Input from './Input';

import ListTodo from './ListTodo';


class Todo extends Component {


state = {

todos: []

}


componentDidMount(){

this.getTodos();

}
```

```
getTodos = () => {

axios.get('/api/todos')

.then(res => {

if(res.data){

this.setState({

todos: res.data

})

}

})

.catch(err => console.log(err))

}



deleteTodo = (id) => {



    axios.delete(`/api/todos/${id}`)

      .then(res => {

        if(res.data){

          this.getTodos()

        }

      })

      .catch(err => console.log(err))
}



render() {

let { todos } = this.state;



    return(

      <div>

        <h1>My Todo(s)</h1>

        <Input getTodos={this.getTodos}/>

        <ListTodo todos={todos} deleteTodo={this.deleteTodo}/>

      </div>

    )



}

}



export default Todo;
```

Goto src folder:

Cd ..

```
vi App.js
import React from 'react';
import Todo from './components/Todo';

import './App.css';



const App = () => {

return (

<div className="App">

<Todo />

</div>

);

}



export default App;


vi App.css
.App {

text-align: center;

font-size: calc(10px + 2vmin);

width: 60%;

margin-left: auto;

margin-right: auto;

}



input {
```

```
height: 40px;

width: 50%;

border: none;

border-bottom: 2px #101113 solid;

background: none;

font-size: 1.5rem;

color: #787a80;

}


input:focus {

outline: none;

}


button {

width: 25%;

height: 45px;

border: none;

margin-left: 10px;

font-size: 25px;

background: #101113;

border-radius: 5px;

color: #787a80;
```

```
cursor: pointer;

}


button:focus {

outline: none;

}


ul {

list-style: none;

text-align: left;

padding: 15px;

background: #171a1f;

border-radius: 5px;

}


li {

padding: 15px;

font-size: 1.5rem;

margin-bottom: 15px;

background: #282c34;

border-radius: 5px;

overflow-wrap: break-word;
```

```
cursor: pointer;

}



@media only screen and (min-width: 300px) {

.App {

width: 80%;

}



input {

width: 100%

}



button {

width: 100%;

margin-top: 15px;

margin-left: 0;

}

}



@media only screen and (min-width: 640px) {

.App {

width: 60%;
}

input {

width: 50%;

}

button {

width: 30%;

margin-left: 10px;

margin-top: 0;

}

}

vim index.css
body {

margin: 0;

padding: 0;

font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", "Roboto",
"Oxygen",

"Ubuntu", "Cantarell", "Fira Sans", "Droid Sans", "Helvetica Neue",

sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;
```

```
box-sizing: border-box;

background-color: #282c34;

color: #787a80;

}


code {

font-family: source-code-pro, Menlo, Monaco, Consolas, "Courier New",

monospace;

}
```

Then go to Todo directory and run:

```
npm run dev
```

.