AWS MERN WEB STACK IMPLEMENTATION

This project shows how to implement a web solution on MERN stack in AWS Cloud, MERN stands for (MongoDB, ExpressJS, ReactJS, Node.js,)

MERN Web stack consists of the following components:

MongoDB: A document-based, No-SQL database used to store application data in the form of documents.

ExpressJS: A server-side Web Application framework for Node.js.

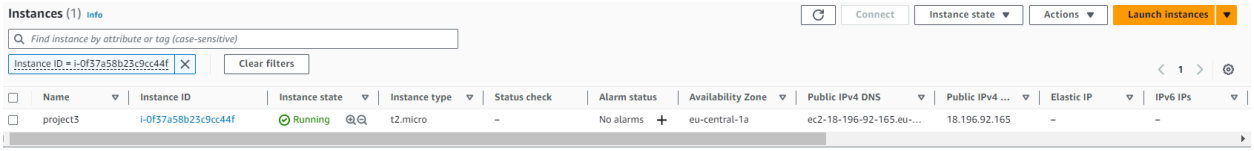
ReactJS: A frontend framework developed by Facebook. It is based on JavaScript, used to build User Interface (UI) components.

Node.js: A JavaScript runtime environment. It is used to run JavaScript on a machine rather than in a browser.

SIMPLE TO-DO APPLICATION ON MERN WEB STACK

STEP 1 – BACKEND CONFIGURATION

We log on to AWS Cloud Services and create an EC2 Ubuntu Instance:



Using the folowing scripts below, we would install the dependencies and Install node.js

sudo apt update

sudo apt upgrade

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

Install NodeJs

sudo apt-get install -y nodejs

Verify the node installation with the command below

node -v

npm -v (sudo apt install npm) to install the npm

```
ubuntu@ip-172-31-26-200:~$ npm -v
8.5.1
ubuntu@ip-172-31-26-200:~$ node -v
v12.22.9
ubuntu@ip-172-31-26-200:~$
```

Create a new directory for your To-Do project:

mkdir Todo

Run the command below to verify that the Todo directory is created with the ls command.

ls

Now change your current directory to the newly created one:

cd Todo

Next, you will use the command npm init to initialise your project, so that a new file named package.json will be created. This file will normally contain information about your application and the dependencies that it needs to run. Follow the prompts after running the command. You can press Enter several times to accept default values.

npm init

Run the command ls to confirm that you have package.json file created.

```
Is this OK? (yes) yes
ubuntu@ip-172-31-26-200:~/Todo$ ls
package.json
ubuntu@ip-172-31-26-200:~/Todo$
```

Step 2: INSTALL EXPRESSJS

To use express, install it using npm:

npm install express

Now create a file index.js with the command below:

touch index.js

Run ls to confirm that your index.js file is successfully created and Install the dotenv module:

npm install dotenv

Open the index.js file with the command below

vim index.js

Then simply paste the code below into the file, which is specified to port 5000, which we will enable us access the index from the browser

```
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}`)
});
```

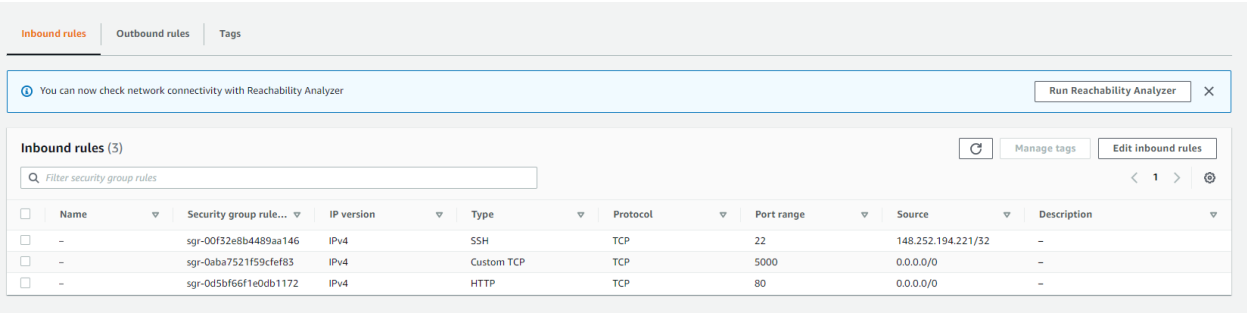
Using the **cat index.js** we can see the code was correctly uploaded as shown below

```
ubuntu@ip-172-31-26-200:~/Todo$ cat index.js
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}`)
});
```

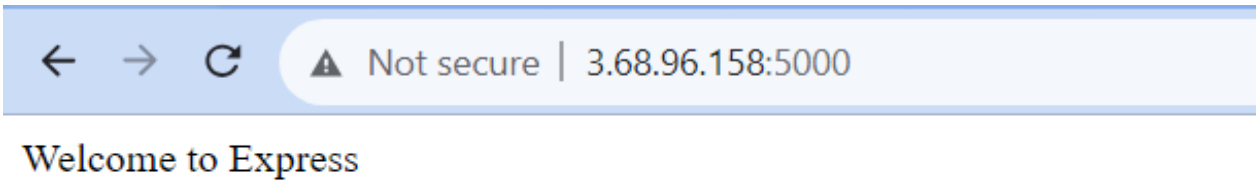
After the code has been inputted, it is time to start our server to see if it works. Open your terminal in the same directory as your index.js file and type: **node index.js**

```
ubuntu@ip-172-31-26-200:~/Todo$ node index.js
Server running on port 5000
```

Next Up I went to my AWS console and created an inbound rule to open TCP port 80, you need to do the same for port 5000, like this:



Then go to access your server's Public IP or Public DNS name followed by port 5000:
http://<PublicIP-or-PublicDNS>:5000



Welcome to Express

Step 3 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.

Creation of a Schema and a model, install mongoose which is a Node.js package that makes working with mongodb easier.

Change directory back Todo folder with cd .. and install Mongoose

npm install mongoose

Create a new folder models :

mkdir models

Change directory into the newly created 'models' folder with

cd models

Inside the models folder, create a file and name it todo.js

touch todo.js

Open the file created with vim todo.js then paste the code below in the file:

```
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;
```

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

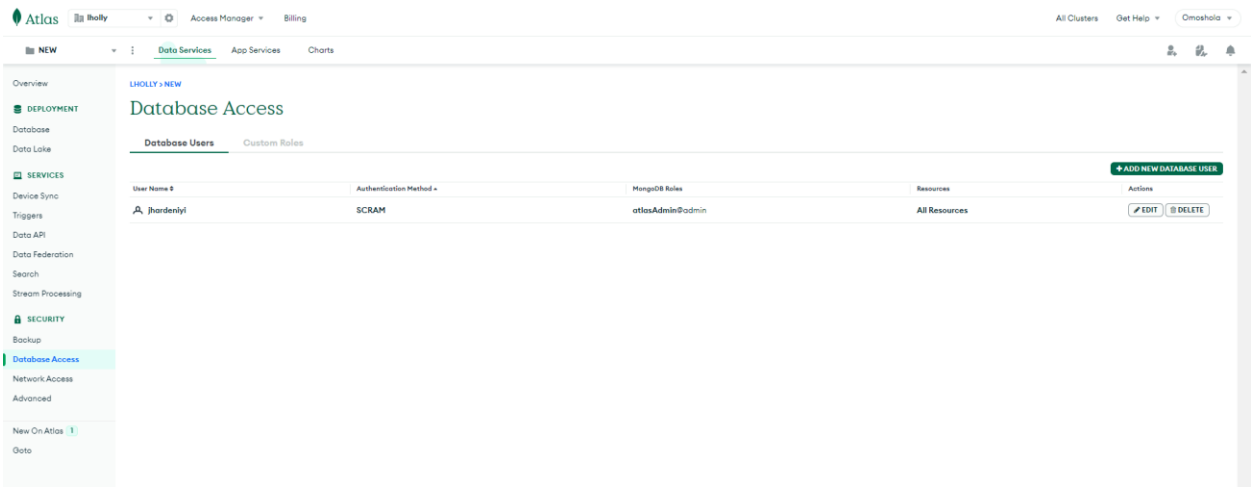
```
const express = require ('express');
const router = express.Router();
const Todo = require('../models/todo');
router.get('/todos', (req, res, next) => {
//this will return all the data, exposing only the id and action field to the client
Todo.find({}, 'action')
.then(data => res.json(data))
.catch(next)
});
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;
```

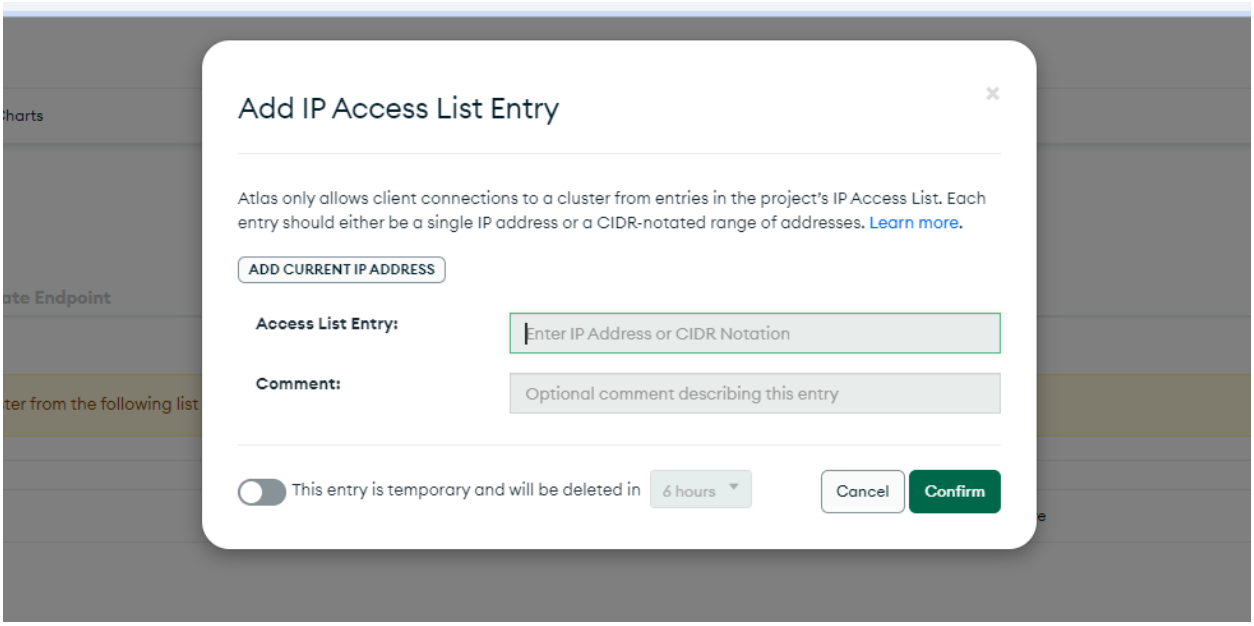
MONGODB DATABASE

We will need a database to store all information when we make a post request to an endpoint. We will be using mLab which provides a DBaaS (Database as a service) solution.

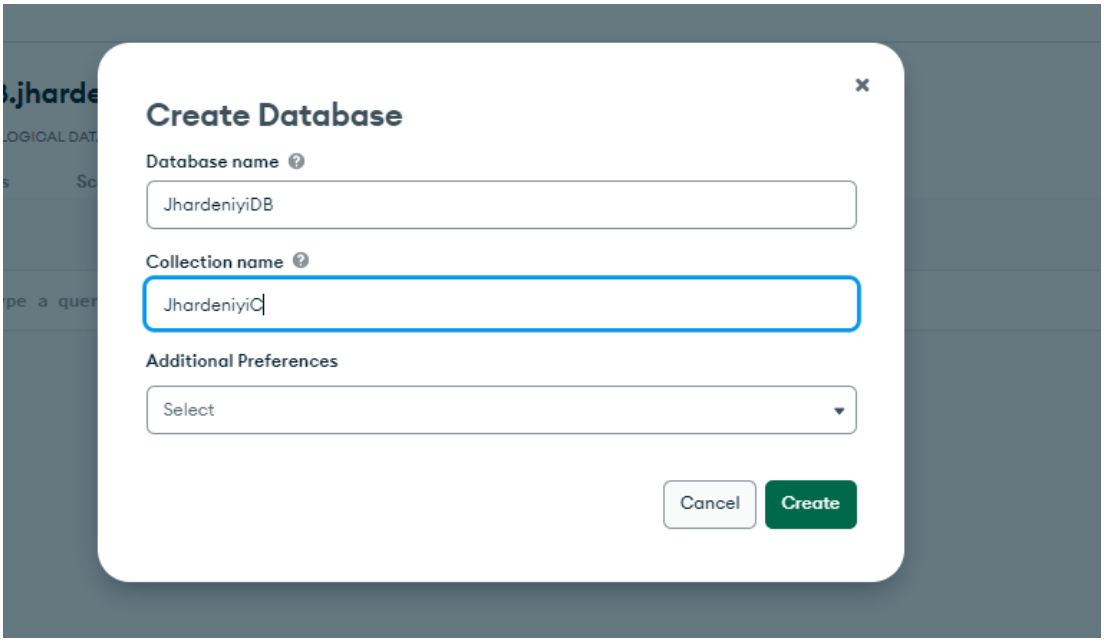
For this we will make use of mLab which provides MongoDB as a service solution to create a cluster.



Next, we add an IP Address choosing Anywhere which we are just using for testing



Once IP address has been created go back to the cluster created and click on collections and select **Add my own data**



Create a file in your Todo directory and name it .env

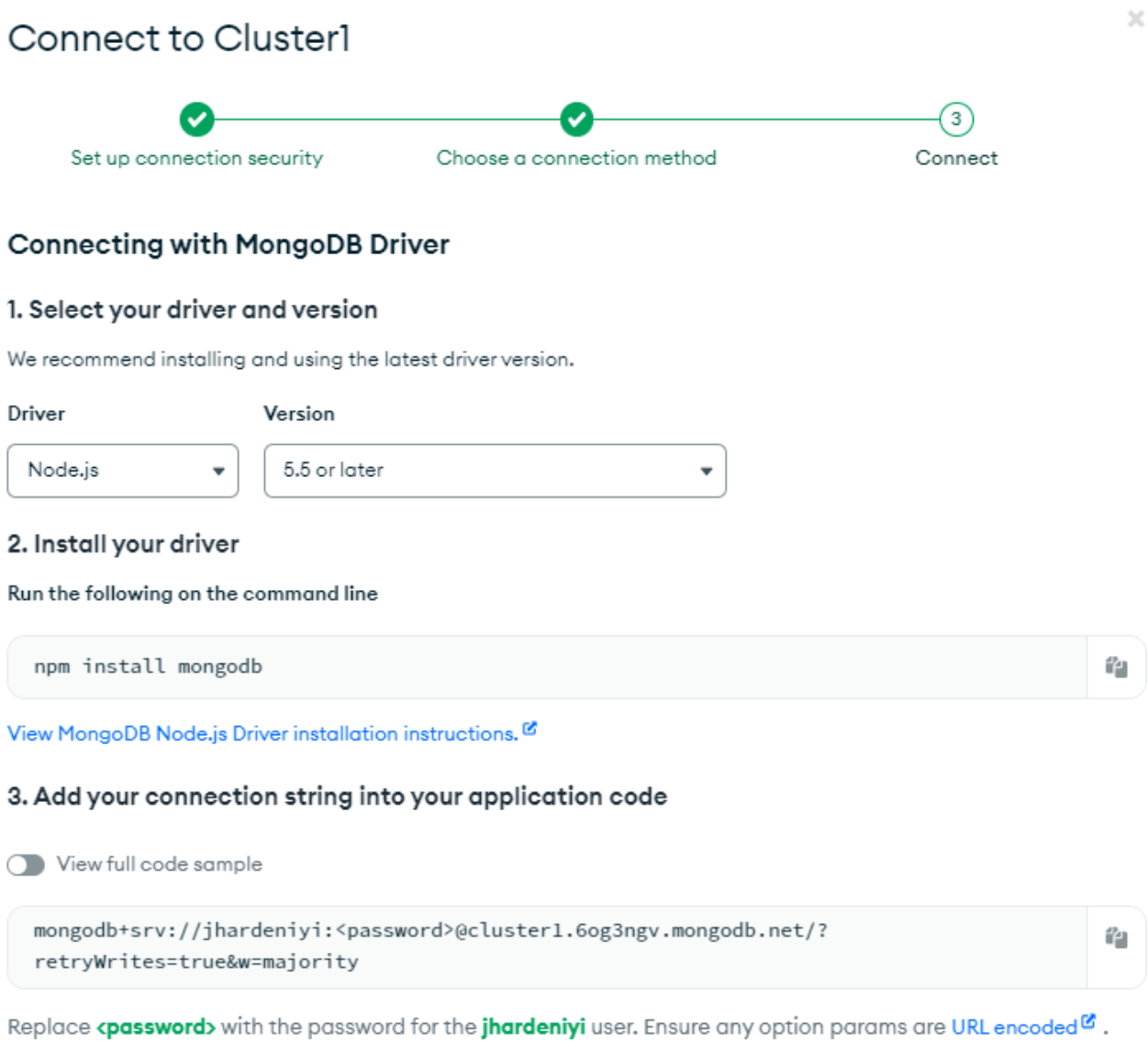
touch .env

vi .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'

This can be obtained from mongoDB as shown below :



Now we need to update the index.js to reflect the use of .env so that Node.js can connect to the database by opening the existing index.js and deleting the content using %d, once that is done paste the 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}`)
});
```

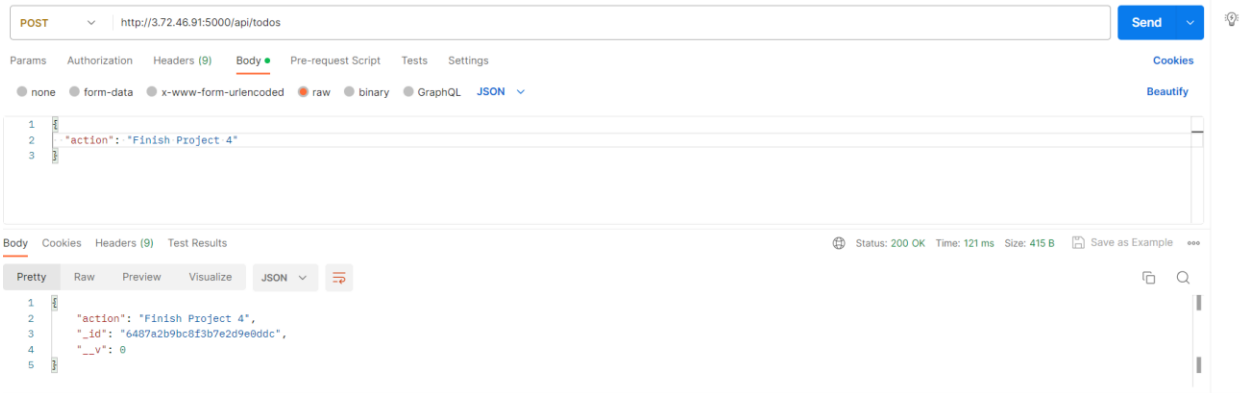
Start your server using the command:

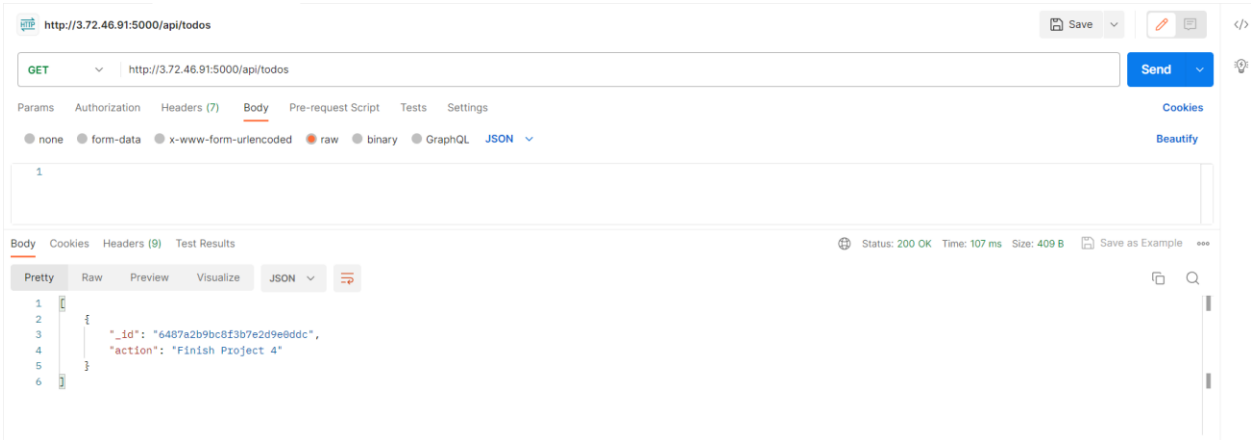
node index.js

Testing Backend Code Using Postman

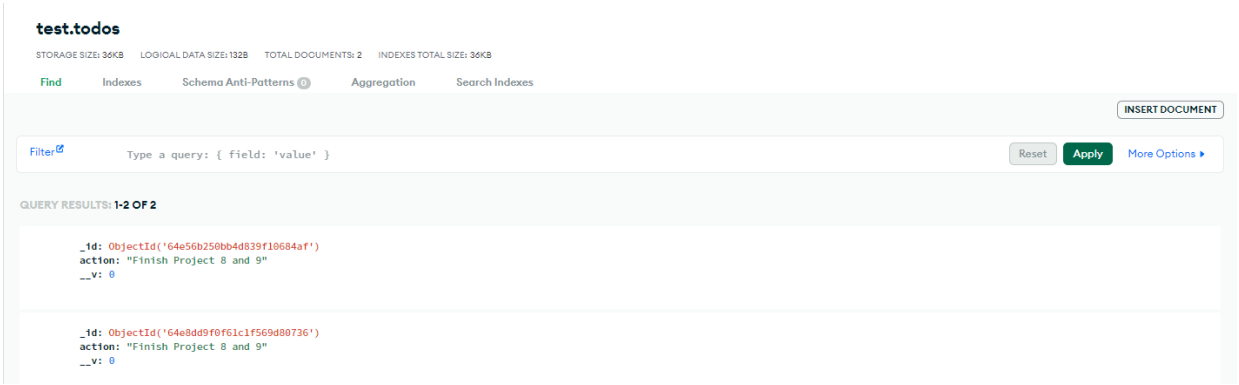
So far, we have built the backend of our application and in order to test to see if it works without a frontend, we use postman to test the endpoints.

On Postman, we make a POST request to our database whilst specifying an action in the body of the request.





Then We make a GET request to see if we can get back what has been posted into the database.



Creating Frontend

After successfully creating the functionality of 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 front end 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

```
ubuntu@ip-172-31-19-206:~/Todo$  npx create-react-app client
Need to install the following packages:
  create-react-app@5.0.1
Ok to proceed? (y) y
npm WARN deprecated tar@2.2.2: This version of tar is no longer supported, and will not receive security updat
es. Please upgrade asap.

Creating a new React app in /home/ubuntu/Todo/client.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...


added 1439 packages in 1m

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

Initialized a git repository.

Installing template dependencies using npm...
```

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

```
ubuntu@ip-172-31-19-206:~/Todo$ npm install concurrently --save-dev

added 30 packages, and audited 114 packages in 13s

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

found 0 vulnerabilities
ubuntu@ip-172-31-19-206:~/Todo$ npm install nodemon --save-dev

added 33 packages, and audited 147 packages in 1s

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

3. 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\""
   },

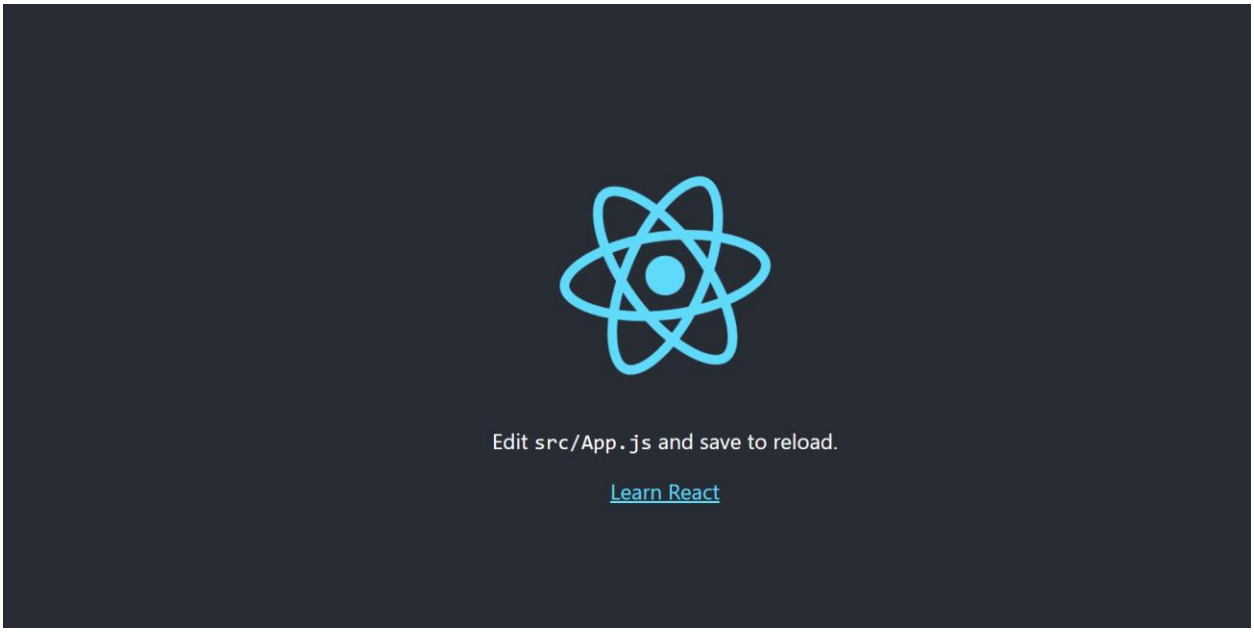
Configure Proxy in package.json and Open the package.json file

cd client

vi package.json

Important note: In order to be able to access the application from the Internet you have to open TCP port 3000 on EC2 by adding a new Security Group rule.

Now, ensure you are inside the Todo directory, and simply do:
npm run dev



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

move to the src directory

cd src

Inside your src folder create another folder called components

mkdir components

Move into the components directory with

cd components

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

touch Input.js ListTodo.js Todo.js

Open Input.js file

vi Input.js

Copy and paste the following

```
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.

Move to the src folder

```
cd ..
```

Move to clients folder

```
cd ..
```

Install Axios

```
npm install axios
```

Go to ‘components’ directory

```
cd src/components
```

After that open your ListTodo.js

```
vi ListTodo.js
```

in the ListTodo.js copy and paste the following code

```
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
```

Then in your Todo.js file you write the following code

```
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;
```

We need to make little adjustment to our react code. Delete the logo and adjust our App.js to look like this.

Move to the src folder

```
cd ..
```

Make sure that you are in the src folder and run

```
vi App.js
```

Copy and paste the code below into it

```
import React from 'react';

import Todo from './components/Todo';
import './App.css';

const App = () => {
return (
```

```
<div className="App">
<Todo />
</div>
);
}

export default App;
```

After pasting, exit the editor.

In the src directory open the App.css

```
vi App.css
```

Then paste the following code into 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;
}
}
```

Exit

In the src directory open the index.css

vim index.css

Copy and paste the code below:

```
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;
}
```

Go to the Todo directory

cd ../..

When you are in the Todo directory run:
npm run dev

Assuming no errors when saving all these files, our To-Do app should be ready and fully functional with the functionality discussed earlier: creating a task, deleting a task, and viewing all your tasks.