**Projects**

* [Projects](https://darey.io/project-listings/)
* [Docs](https://www.darey.io/projects/)
* [(STEP 5) PROJECT 3: MERN STACK IMPLEMENTATION](https://www.darey.io/docs/simple-to-do-application-on-mern-web-stack/)

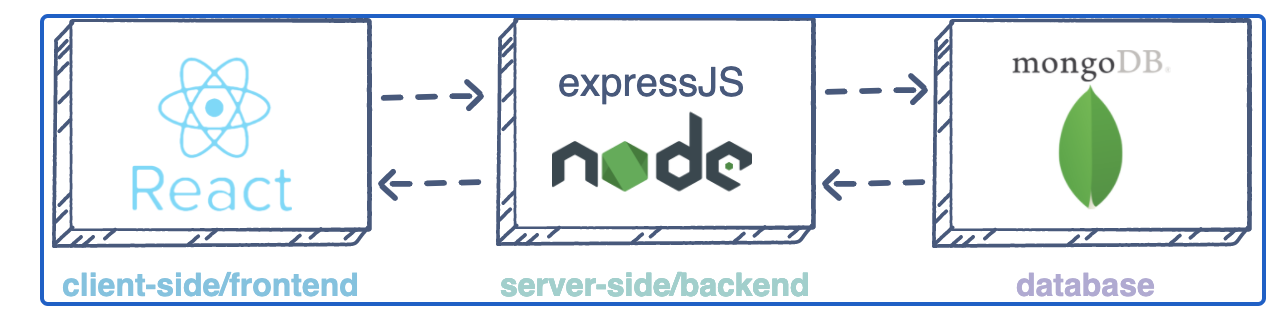
**SIMPLE TO-DO APPLICATION ON MERN WEB STACK**

 CompletedSubmit Project for Review

In this project, you are tasked to implement a web solution based on **MERN** stack in AWS Cloud.

**MERN** Web stack consists of following components:

1. [**M**ongoDB](https://www.mongodb.com/): A document-based, No-SQL database used to store application data in a form of documents.
2. [**E**xpressJS](https://expressjs.com/): A server side Web Application framework for Node.js.
3. [**R**eactJS](https://reactjs.org/): A frontend framework developed by Facebook. It is based on [JavaScript](https://www.javascript.com/), used to build User Interface (UI) components.
4. [**N**ode.js](https://nodejs.org/en/): A JavaScript runtime environment. It is used to run JavaScript on a machine rather than in a browser.



As shown on the illustration above, a user interacts with the ReactJS UI components at the application front-end residing in the browser. This frontend is served by the application backend residing in a server, through ExpressJS running on top of NodeJS.

Any interaction that causes a data change request is sent to the NodeJS based Express server, which grabs data from the MongoDB database if required, and returns the data to the frontend of the application, which is then presented to the user.

Side Self Study

1. Make a research what types of [Database Management Systems (DBMS) exist and what each type is more suitable for](https://www.alooma.com/blog/types-of-modern-databases). Be able to explain the difference between Relational DBMS and NoSQL (of a different kind).
2. Get yourself familiar with a concept of [Web Application Frameworks](https://en.wikipedia.org/wiki/Web_framework). Get to know what server-side (backend) and client-side (forntend) frameworks exist and what they are used for.
3. [Practice basic JavaScript syntax just for fun](https://www.w3schools.com/js/js_intro.asp).
4. Explore what [RESTful API](https://restfulapi.net/) is and what it is used for in Web development.
5. Read what [Cascading Style Sheets (CSS)](https://en.wikipedia.org/wiki/CSS) is used for and browse [basic syntax and properties](https://www.w3schools.com/css/css_intro.asp).

Instructions On How To Submit Your Work For Review And Feedback

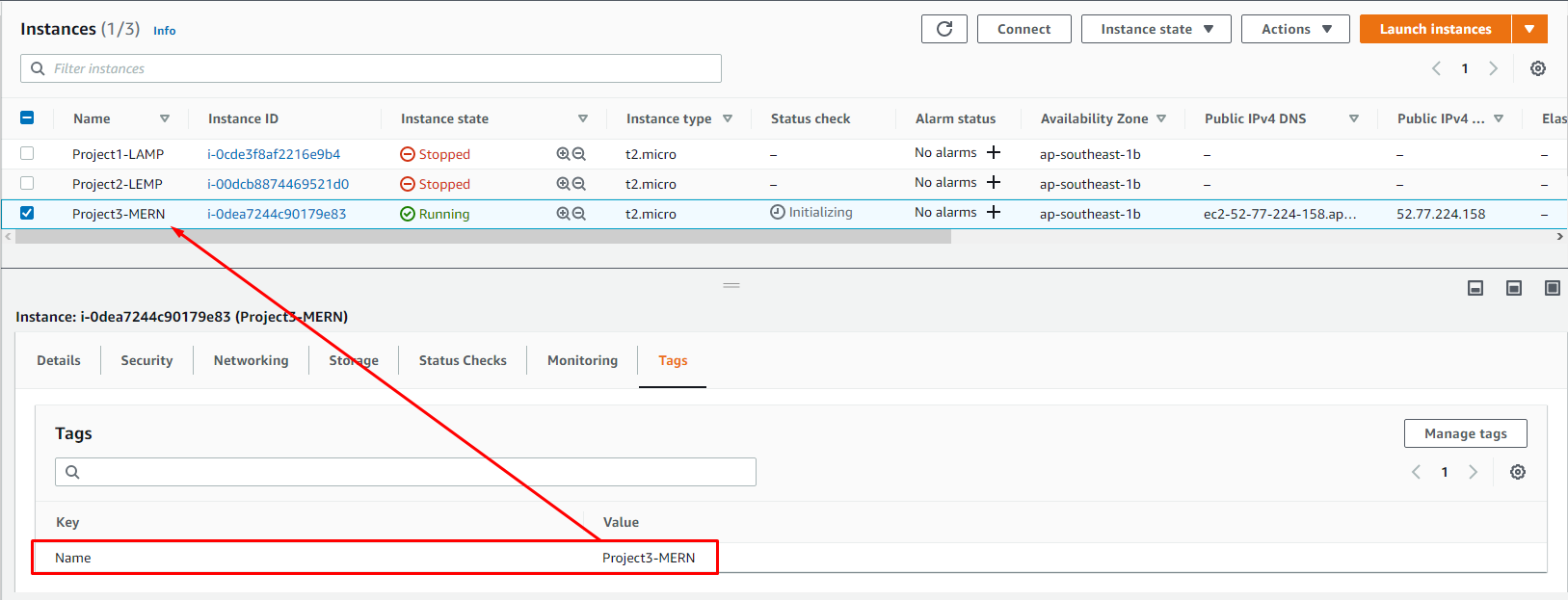
To submit your work for review and feedback – follow [**this instruction**](https://darey.io/docs/how-to-submit-your-work-for-review-and-feedback-2/).

Step 0 – Preparing prerequisites

In order to complete this project you will need an AWS account and a virtual server with Ubuntu Server OS.

If you do not have an AWS account – go back to [**Project 1 Step 0**](https://starter-pbl.darey.io/en/latest/project1.html) to sign in to AWS free tier account and create a new EC2 Instance of t2.nano family with Ubuntu Server 20.04 LTS (HVM) image. Remember, you can have multiple EC2 instances, but make sure you **STOP** the ones you are not working with at the moment to save available free hours.

**Hint #1:** When you create your EC2 Instances, you can add Tag "Name" to it with a value that corresponds to a current project you are working on – it will be reflected in the name of the EC2 Instance. Like this:



**Hint #2 (for Windows users only):** In previous projects we used Putty and Git Bash to connect to our EC2 Instances.

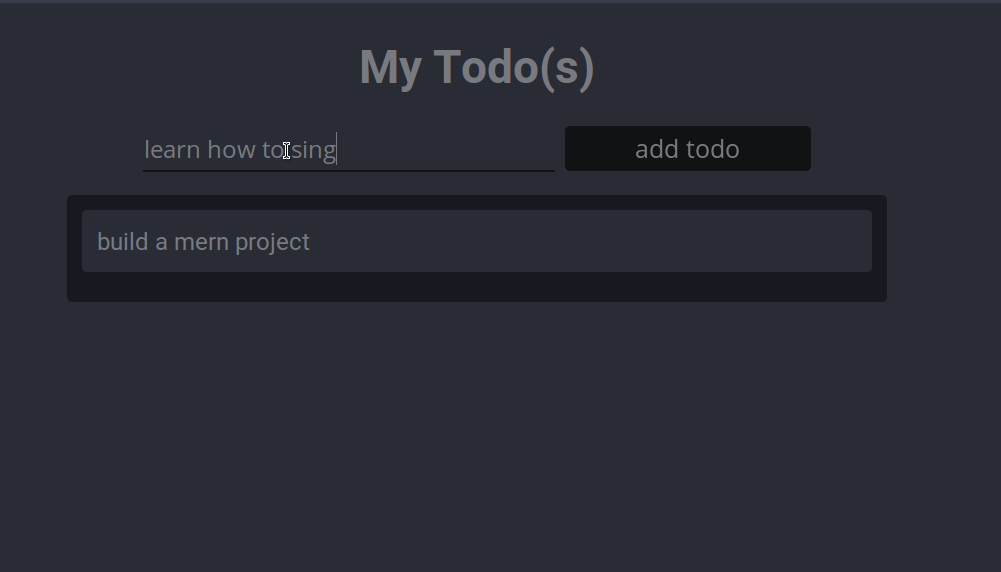
In this project and going forward, we are going to explore the usage [windows terminal](https://docs.microsoft.com/en-us/windows/terminal/install).

Watch this videos to learn how to set up windows terminal on your pc;

* [windows installatatiosn part 1](https://youtu.be/R-qcpehB5HY)
* [windows installatatiosn part 2](https://youtu.be/jsNIlK5s6pI)

Task

To deploy a simple **To-Do** application that creates To-Do lists like this:



# STEP 1 – BACKEND CONFIGURATION

Update ubuntu

sudo apt update

Upgrade ubuntu

sudo apt upgrade

Lets get the location of Node.js software from [Ubuntu repositories](https://github.com/nodesource/distributions#deb).

curl -fsSL https://deb.nodesource.com/setup\_18.x | sudo -E bash -

##### Install Node.js on the server

Install Node.js with the command below

sudo apt-get install -y nodejs

**Note**: The command above installs both nodejs and npm. [NPM](https://www.npmjs.com/) is a package manager for Node like apt for Ubuntu, it is used to install Node modules & packages and to manage dependency conflicts.

Verify the node installation with the command below

node -v

Verify the node installation with the command below

npm -v

##### Application Code Setup

Create a new directory for your To-Do project:

mkdir Todo

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

ls

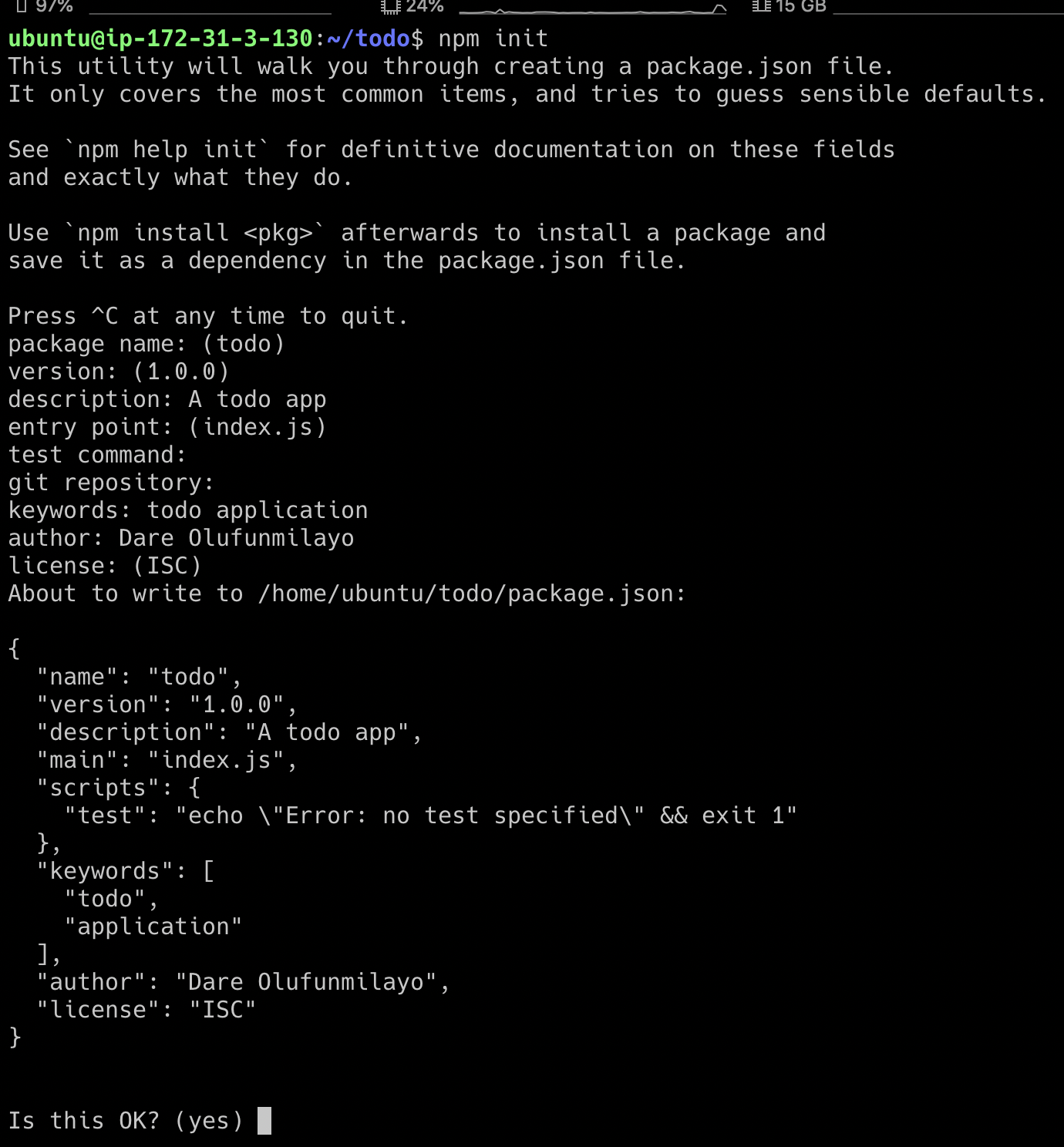
**TIP:** In order to see some more useful information about files and directories, you can use following combination of keys ls -lih – it will show you different properties and size in human readable format. You can learn more about different useful keys for ls command with ls --help.

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, then accept to write out the package.json file by typing yes.

npm init



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

Next, we will Install ExpressJs and create the Routes directory.

# INSTALL EXPRESSJS

##### Install ExpressJS

Remember that Express is a framework for Node.js, therefore a lot of things developers would have programmed is already taken care of out of the box. Therefore it simplifies development, and abstracts a lot of low level details. For example, Express helps to define routes of your application based on HTTP methods and URLs.

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

Install the dotenv module

npm install dotenv

Open the index.js file with the command below

vim index.js

Type the code below into it and save. Do not get overwhelmed by the code you see. For now, simply paste the code into the file.

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}`)

});

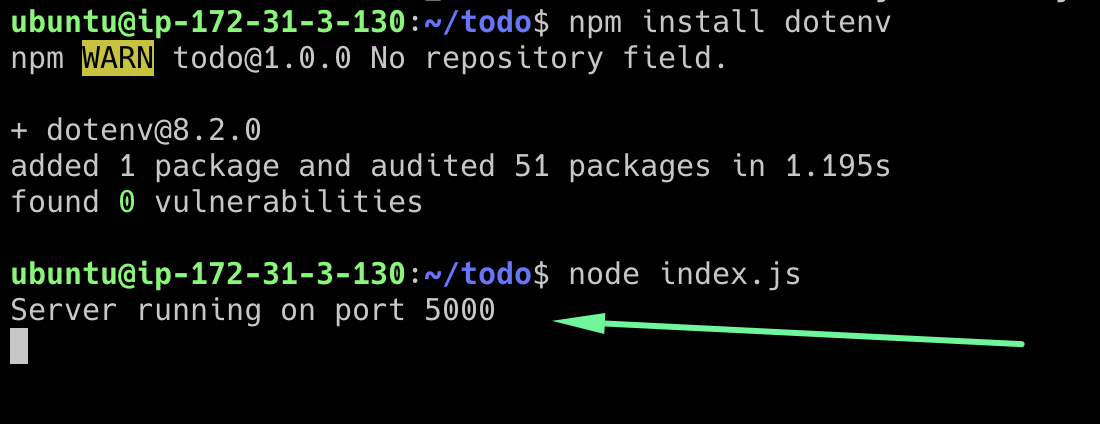
Notice that we have specified to use port 5000 in the code. This will be required later when we go on the browser.

Use :w to save in vim and use :qa to exit vim

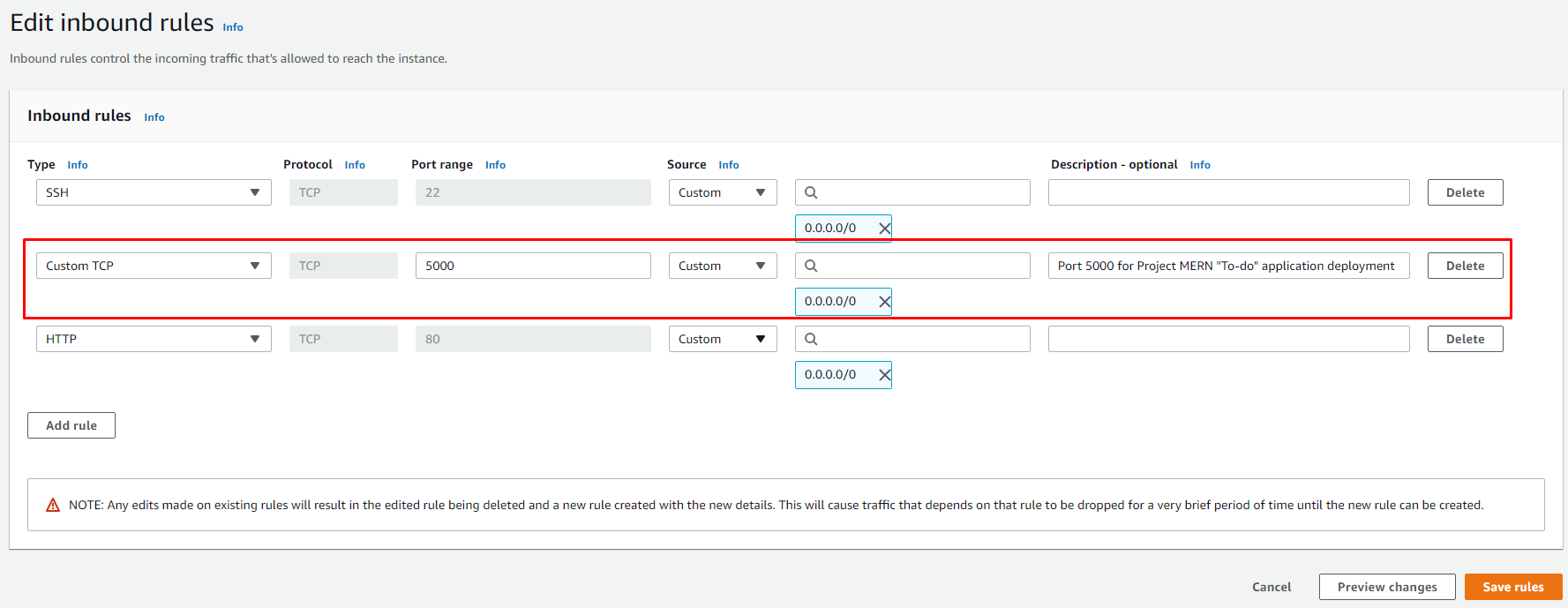
Now it is time to start our server to see if it works. Open your terminal in the same directory as your index.js file and type:

node index.js

If every thing goes well, you should see **Server running on port 5000** in your terminal.



Now we need to open this port in EC2 Security Groups. Refer to [**Project 1**](https://starter-pbl.darey.io/en/latest/project1.html) Step 1 – Installing the Nginx Web Server. There we created an inbound rule to open TCP port 80, you need to do the same for port 5000, like this:

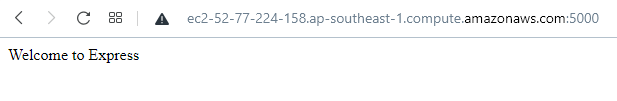


Open up your browser and try to access your server’s Public IP or Public DNS name followed by port 5000:

http://<PublicIP-or-PublicDNS>:5000

Quick reminder how to get your server’s Public IP and public DNS name:  
1) You can find it in your AWS web console in EC2 details  
2) Run curl -s http://169.254.169.254/latest/meta-data/public-ipv4 for Public IP address or curl -s http://169.254.169.254/latest/meta-data/public-hostname for Public DNS name.

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](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods): POST, GET, DELETE.

For each task, we need to create [routes](https://expressjs.com/en/guide/routing.html) that will define various endpoints that the To-do app will depend on. So let us create a folder routes

mkdir routes

**Tip:** You can open multiple shells in Putty or Linux/Mac to connect to the same EC2

Change directory to routes folder.

cd routes

Now, create a file api.js with the command below

touch api.js

Open the file with the command below

vim api.js

Copy below code in the file. (Do not be overwhelmed with the code)

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;

Moving forward let create Models directory.

# MODELS

Now comes the interesting part, since the app is going to make use of [Mongodb](https://www.mongodb.com/" \t "_blank) 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. **(Seems like a lot of information, but not to worry, everything will become clear to you over time. I promise!!!)**

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*](https://mongoosejs.com/) 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

**Tip:** All three commands above can be defined in one line to be executed consequently with help of && operator, like this:

mkdir models && cd models && 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;

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;

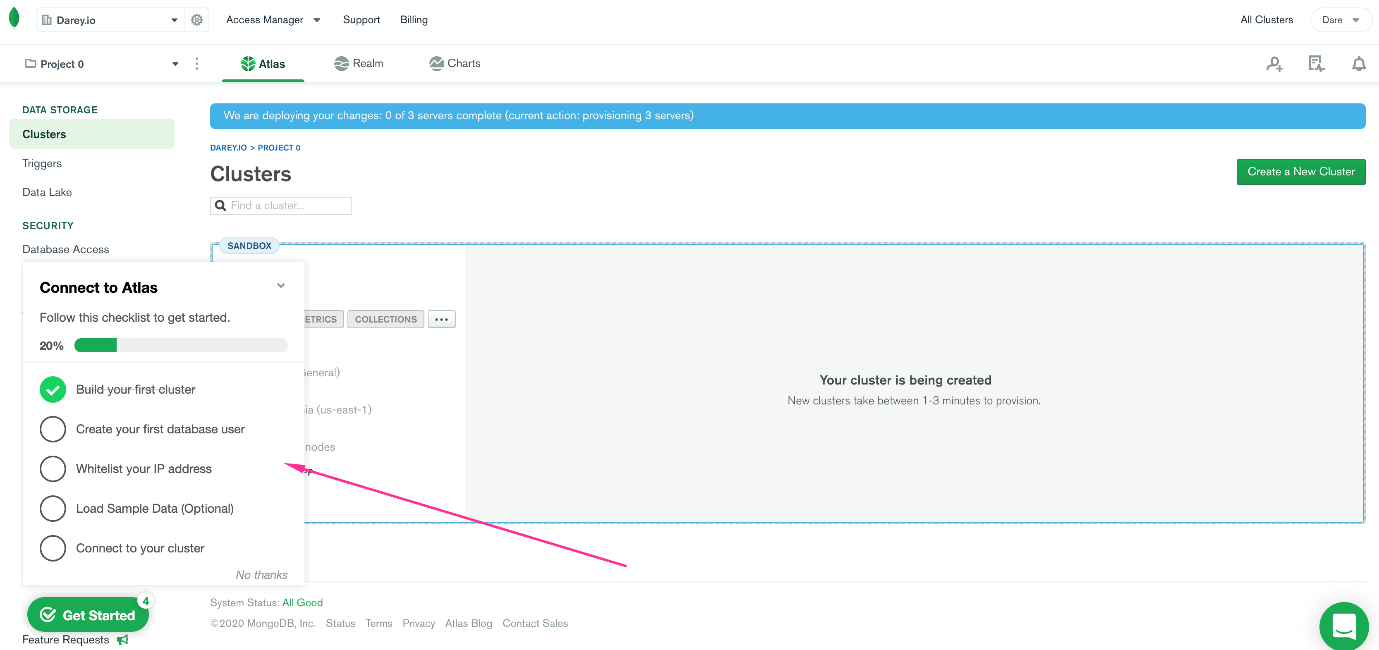
The next piece of our application will be the MongoDB Database

# MONGODB DATABASE

##### MongoDB Database

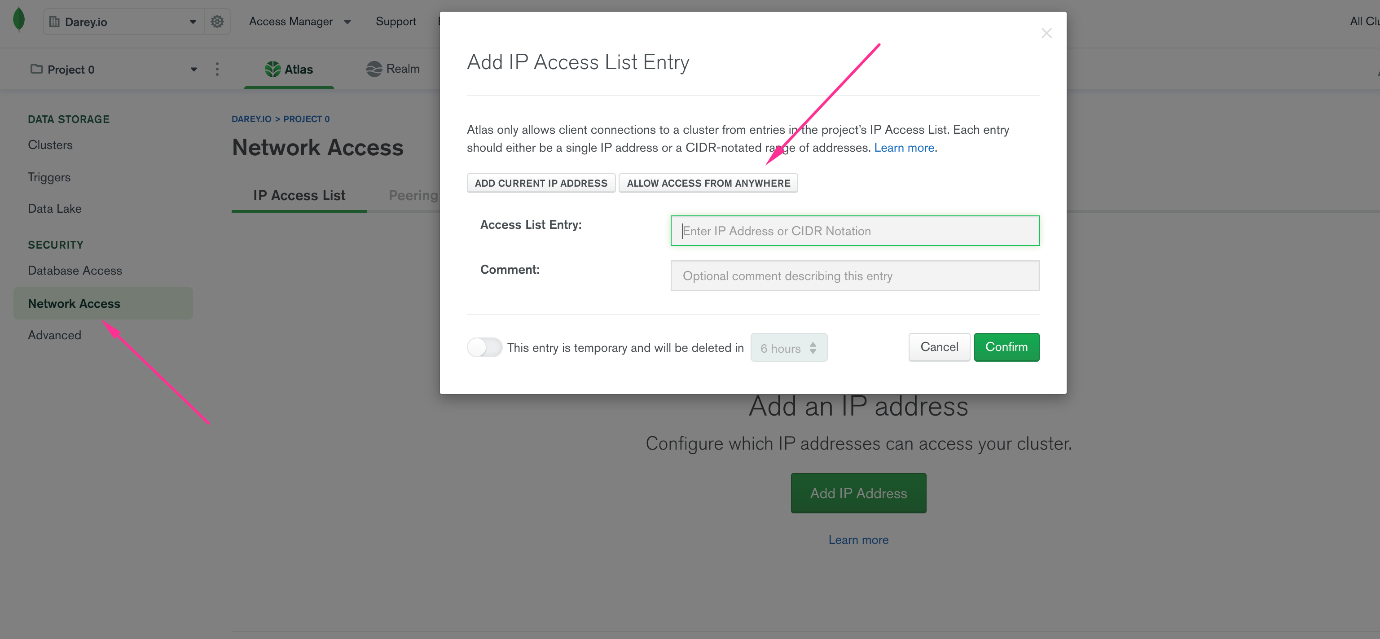
We need a database where we will store our data. For this we will make use of **mLab**. mLab provides MongoDB database as a service solution ([DBaaS](https://en.wikipedia.org/wiki/Cloud_database)), so to make life easy, you will need to sign up for a shared clusters free account, which is ideal for our use case. [Sign up here](https://www.mongodb.com/atlas-signup-from-mlab). Follow the sign up process, select **AWS** as the cloud provider, and choose a region near you.

Complete a get started checklist as shown on the image below

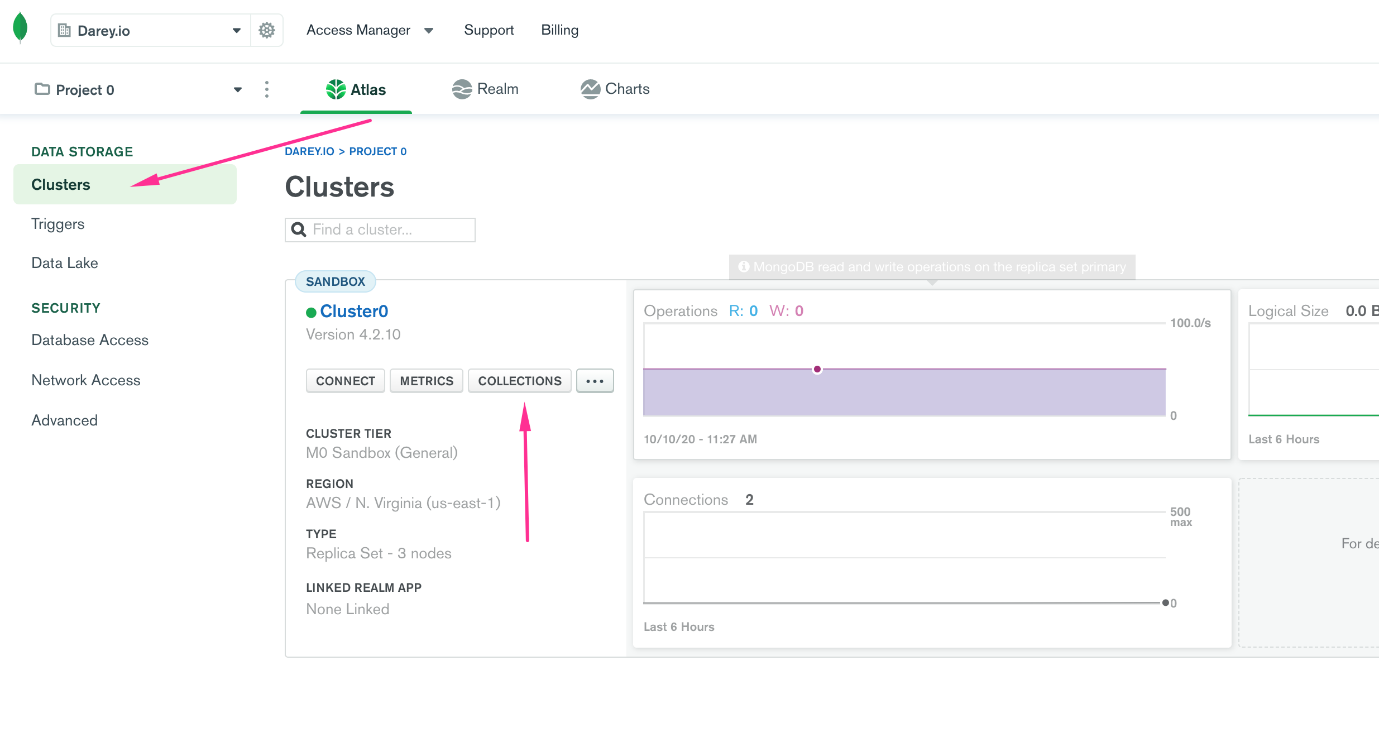


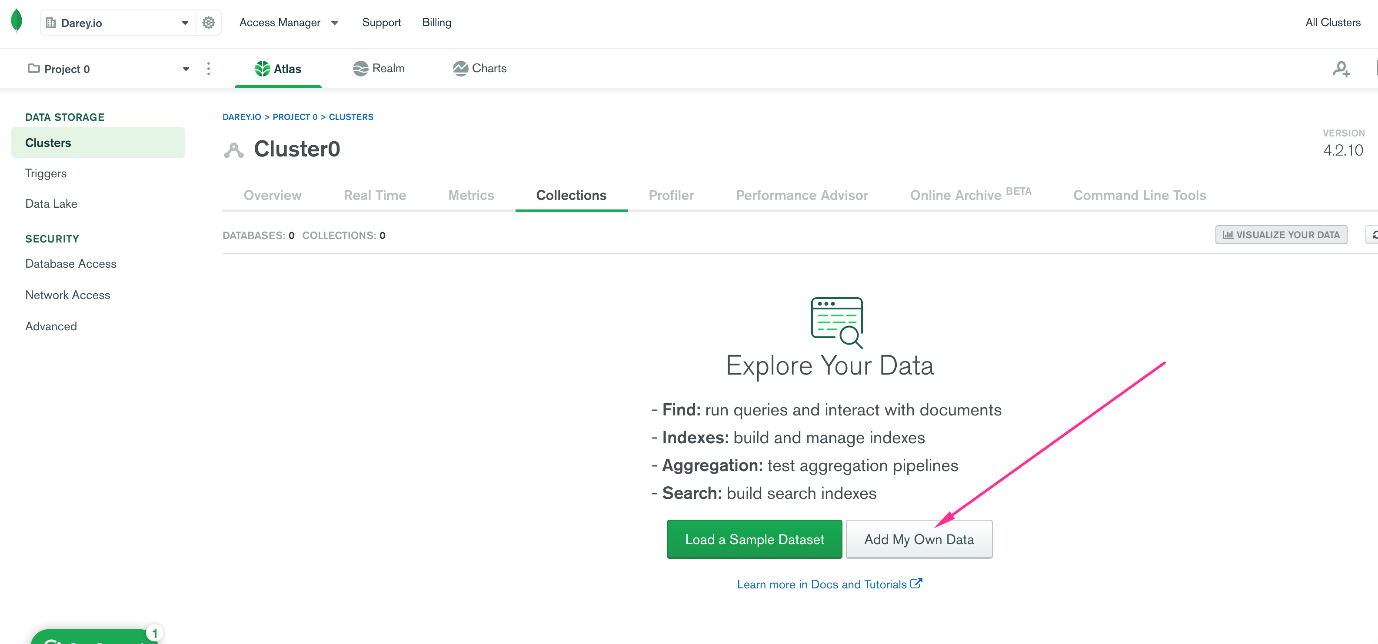
Allow access to the MongoDB database from anywhere (Not secure, but it is ideal for testing)

**IMPORTANT NOTE**  
In the image below, make sure you change the time of deleting the entry from 6 Hours to 1 Week



Create a MongoDB database and collection inside mLab





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.

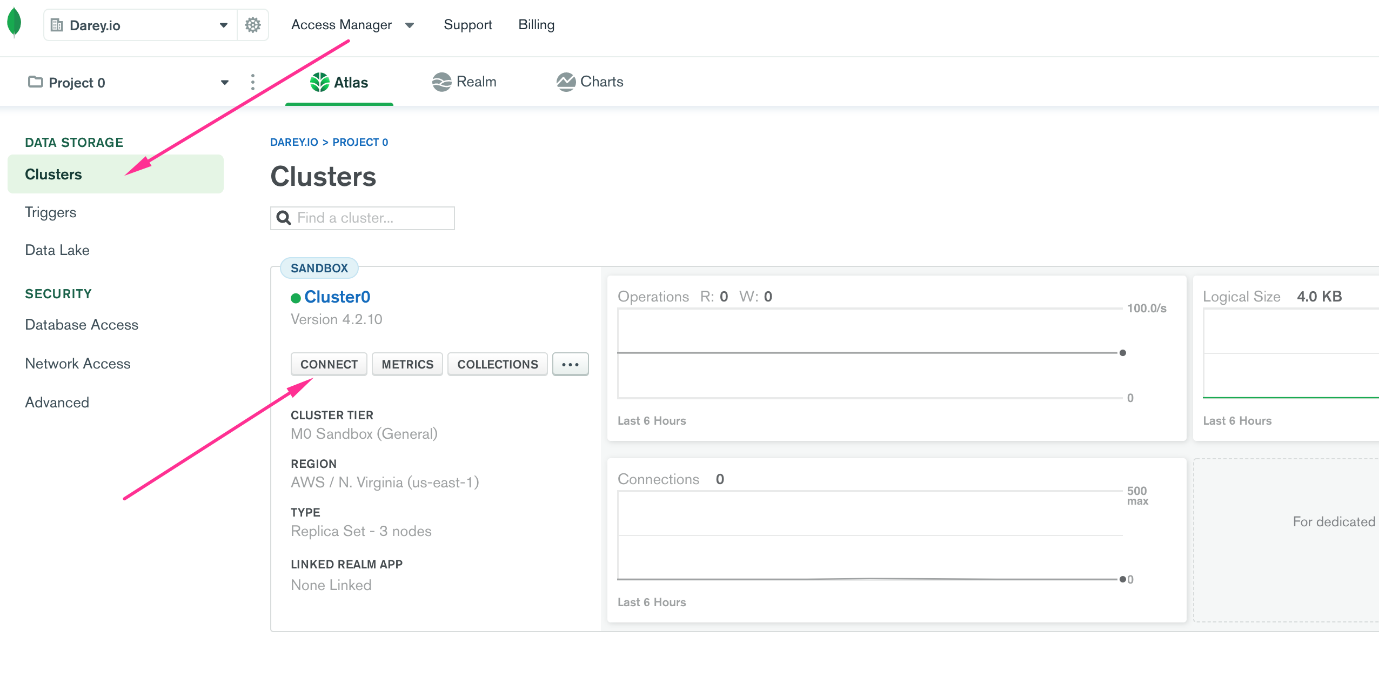
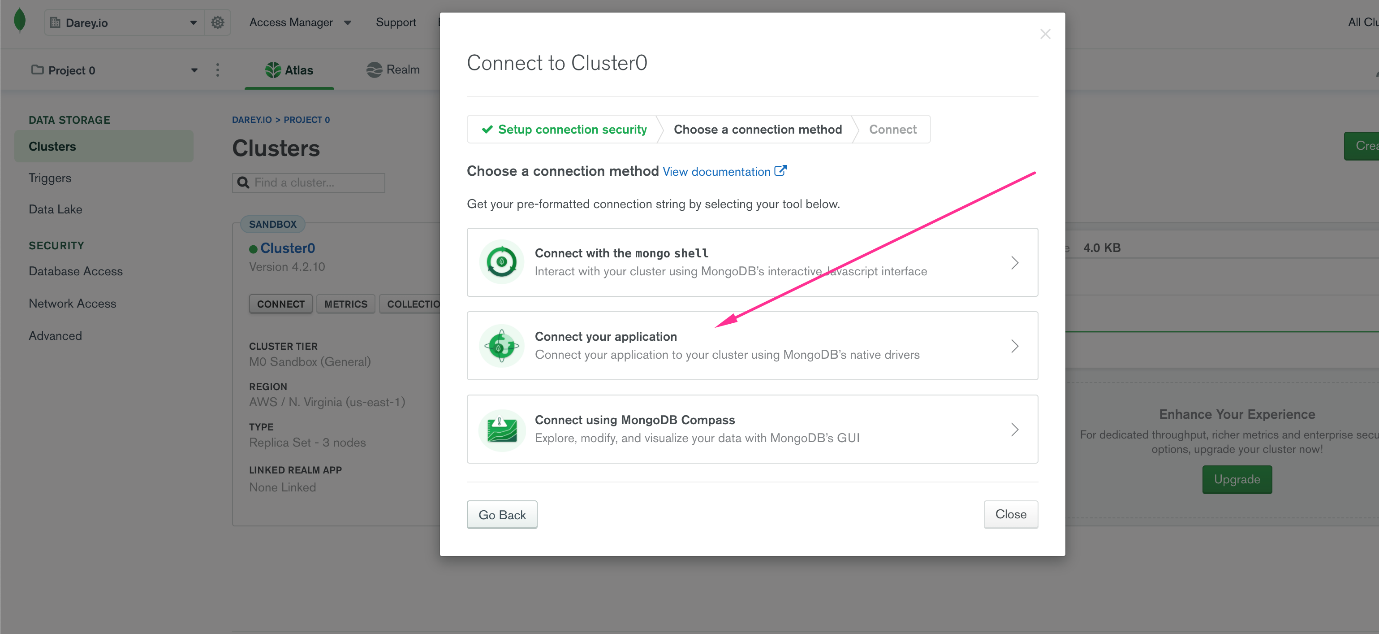
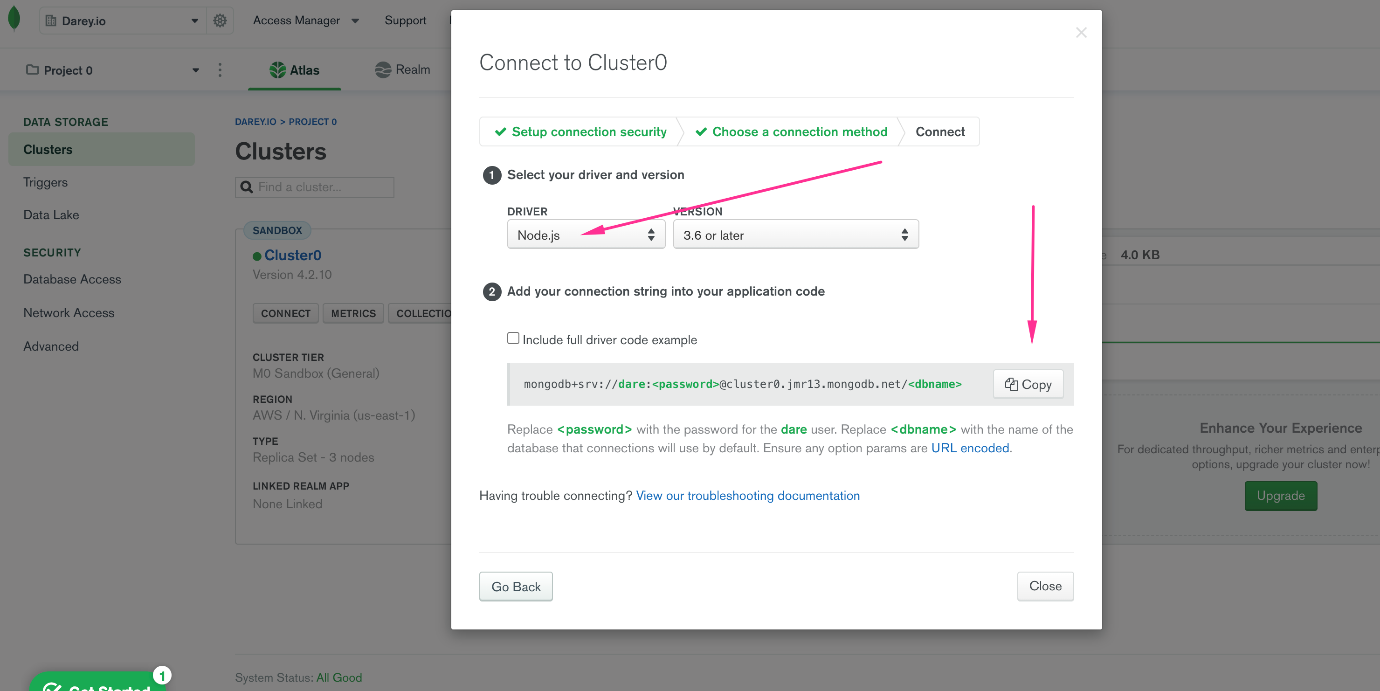
touch .env

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'

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

Here is how to get your connection string  
  
  


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.

To do that using vim, follow below steps

1. Open the file with vim index.js
2. Press esc
3. Type :
4. Type %d
5. Hit ‘Enter’

The entire content will be deleted, then,

1. Press i to enter the insert mode in vim
2. Now, paste the entire code below in the file.

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}`)

});

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.

Start your server using the command:

node index.js

You shall see a message **‘Database connected successfully’**, if so – we have our backend configured. Now we are going to test it.

##### Testing Backend Code without Frontend using RESTful API

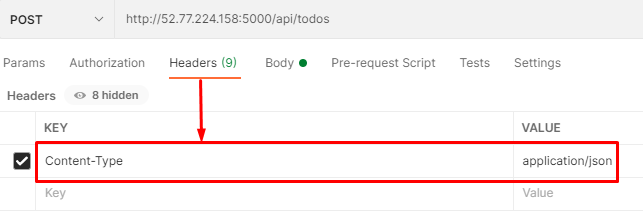
So far we have written backend part of our To-Do application, and configured a database, but we do not have a frontend UI yet. We need ReactJS code to achieve that. But during development, we will need a way to test our code using RESTfulL API. Therefore, we will need to make use of some API development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

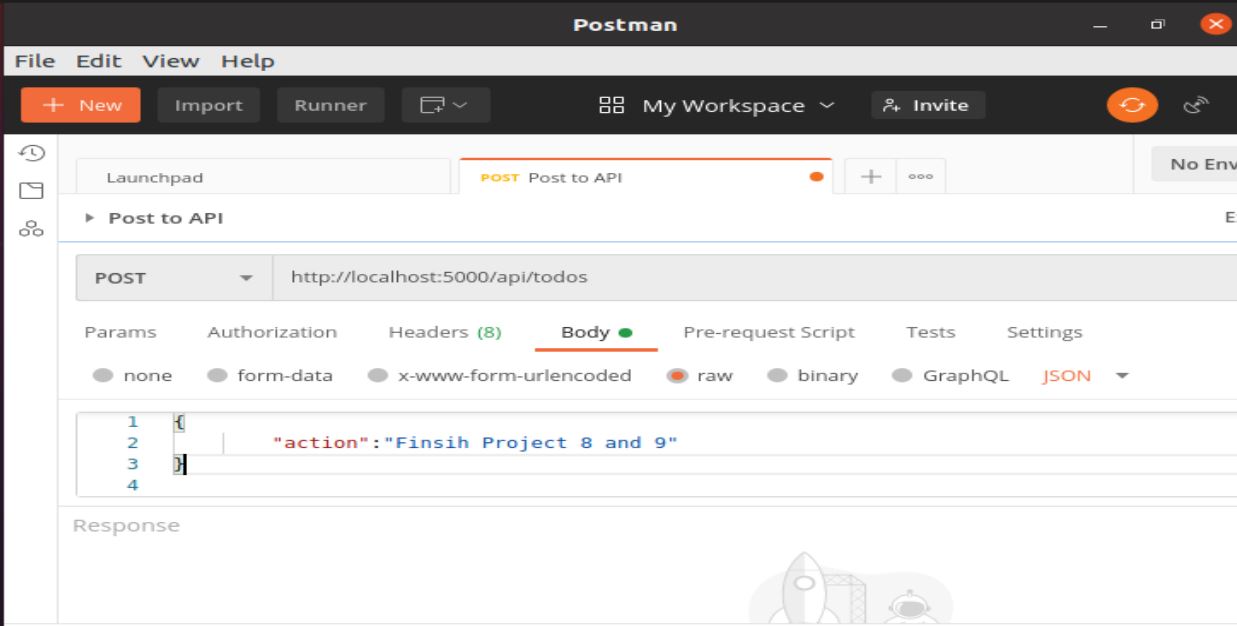
In this project, we will use [Postman](https://www.getpostman.com/) to test our API.  
Click [**Install Postman**](https://www.getpostman.com/downloads/) to download and install postman on your machine.

Click [**HERE**](https://www.youtube.com/watch?v=FjgYtQK_zLE) to learn how perform [CRUD operartions](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete) on Postman

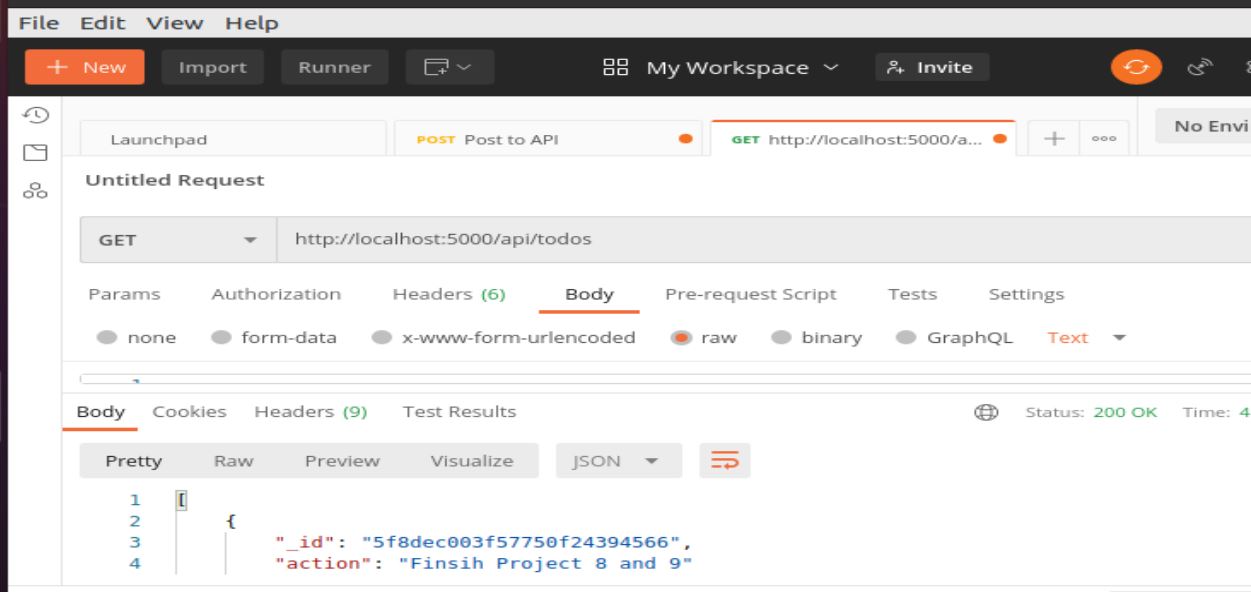
You should test all the API endpoints and make sure they are working. For the endpoints that require body, you should send JSON back with the necessary fields since it’s what we setup in our code.

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  


Check the image below:  


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).



**Optional task:** Try to figure out how to send a DELETE request to delete a task from out To-Do list.

**Hint:** To delete a task – you need to send its ID as a part of DELETE request.

By now you have tested backend part of our To-Do application and have made sure that it supports all three operations we wanted:

* Display a list of tasks – HTTP GET request
* Add a new task to the list – HTTP POST request
* Delete an existing task from the list – HTTP DELETE request

We have successfully created our Backend, now let go create the Frontend.

# STEP 2 – FRONTEND CREATION

#### Step 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](https://www.npmjs.com/package/concurrently). It is used to run more than one command simultaneously from the same terminal window.

npm install concurrently --save-dev

1. Install [nodemon](https://www.npmjs.com/package/nodemon" \t "_blank). 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

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

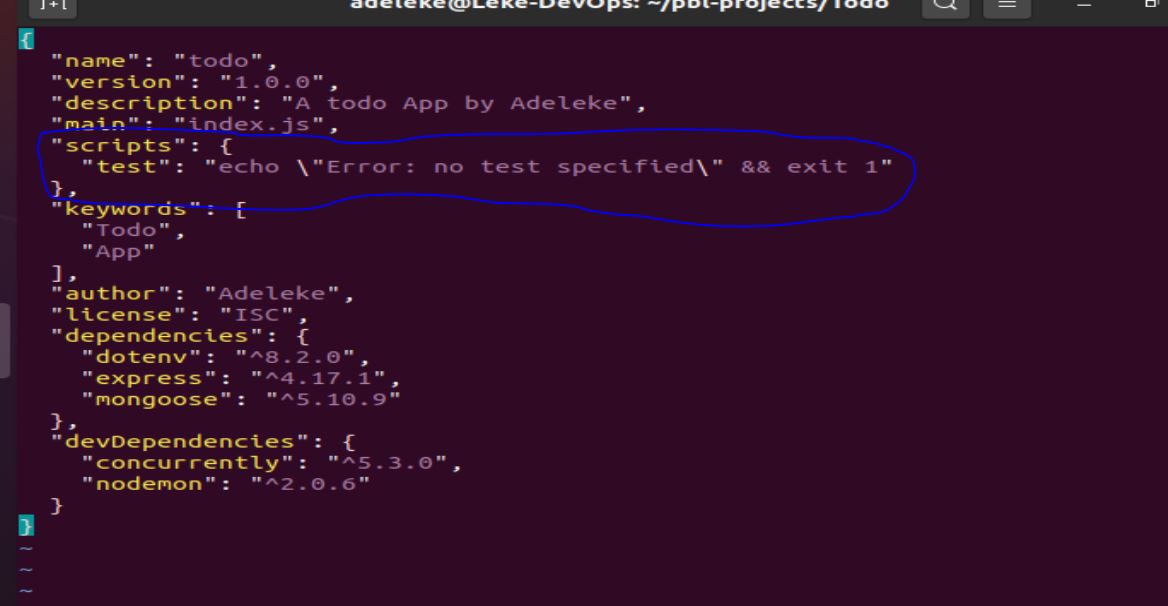
"scripts": {

"start": "node index.js",

"start-watch": "nodemon index.js",

"dev": "concurrently \"npm run start-watch\" \"cd client && npm start\""

},



##### Configure Proxy in package.json

1. Change directory to ‘client’

cd client

1. Open the package.json file

vi package.json

1. 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

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

npm run dev

Your app should open and start running on localhost:3000

**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. You already know how to do it.

##### 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](https://github.com/axios/axios" \t "_blank), 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

Sip a coffee, click on the next button and let finish this up.

**FRONTEND CREATION (CONTINUED)**

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;

}

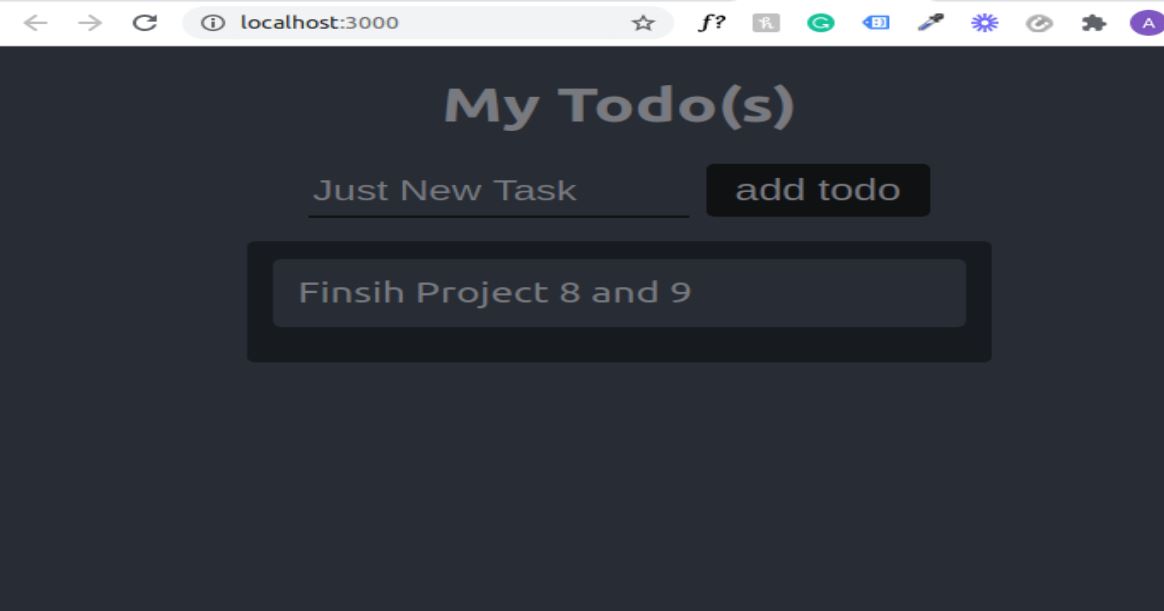
Go to the Todo directory

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.



Congratulations

In this Project #3 you have created a simple To-Do and deployed it to MERN stack. You wrote a frontend application using React.js that communicates with a backend application written using Expressjs. You also created a Mongodb backend for storing tasks in a database.

Credits:

1. [This guide was inspired by Digital Ocean](https://www.digitalocean.com/community/tutorials/getting-started-with-the-mern-stack)
2. [Educative.io](https://www.educative.io/edpresso/what-is-mern-stack)