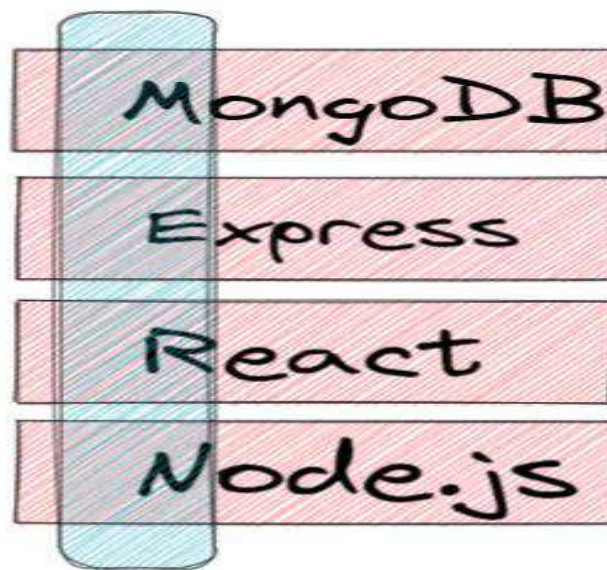


Create Blog website using MERN stack

Introduction

The MERN stack is a popular set of technologies for creating a modern Single Page Application (SPA). MERN stands for MongoDB, Express, React, and Node.js:

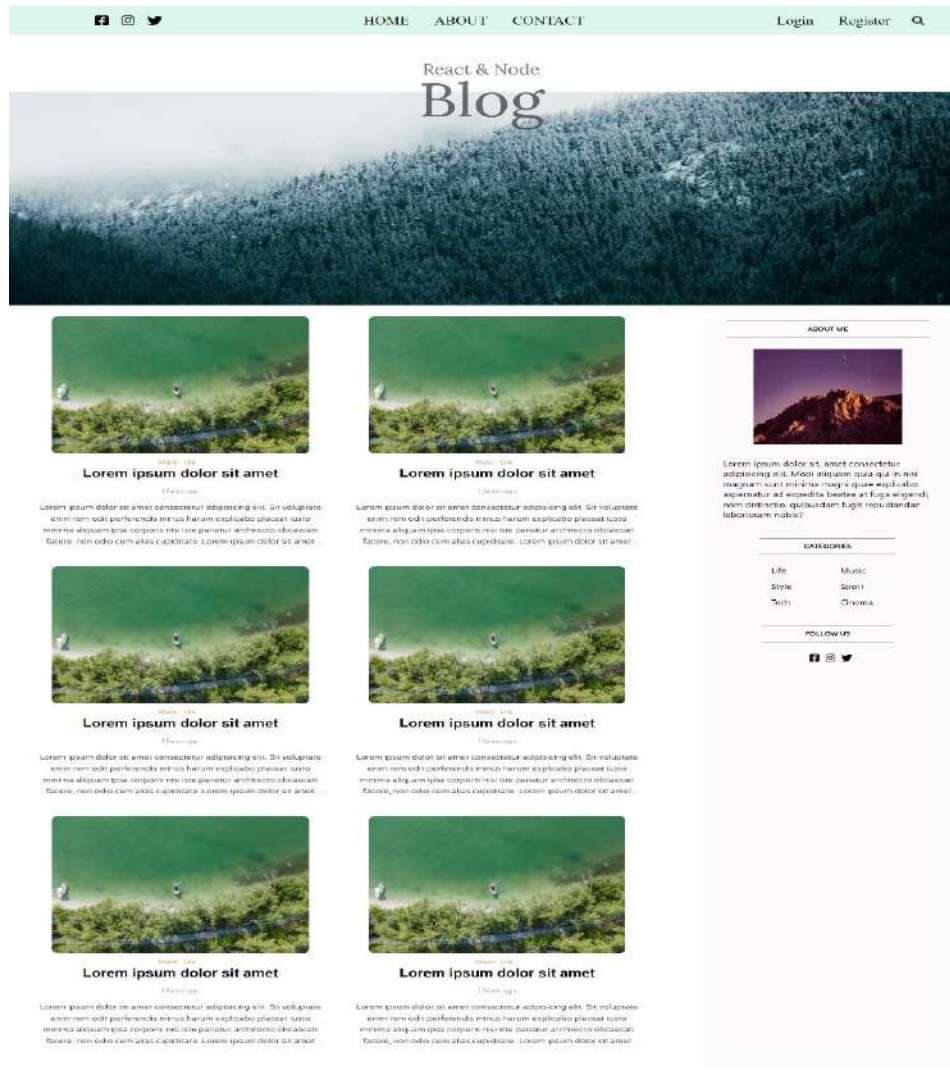
- **Node.js** is a popular server-side framework that allows us to run JavaScript code on a web server.
- **Express** is a Node.js web application framework that makes Node application development simpler and faster.
- **MongoDB** is a NoSQL database that stores data persistently in the form of collections and documents.
- **React** is a JavaScript frontend library for creating user interfaces.



What you will learn

After completing this project, you will be able to create a full-stack blog application that performs CRUD operations by utilizing the MERN stack. This blog tutorial should help you understand the fundamental operations of the MERN stack.

Final Design:



Pre-requisites:

- Basic understanding about HTML , CSS, JSX , React , Node and MongoDB.
- VS code or any other code editor system.
- NodeJS installed on your system.

How to download and install VS code

<https://code.visualstudio.com/download>

How to Download and Install NodeJS

<https://nodejs.org/en/download/>

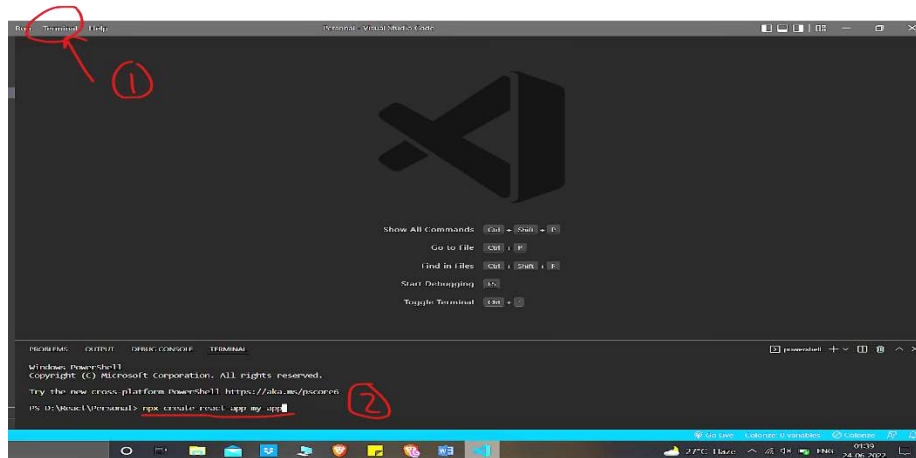
Implementation Step

Firstly we will create front-end , and back-end and database afterwards.

Create react app

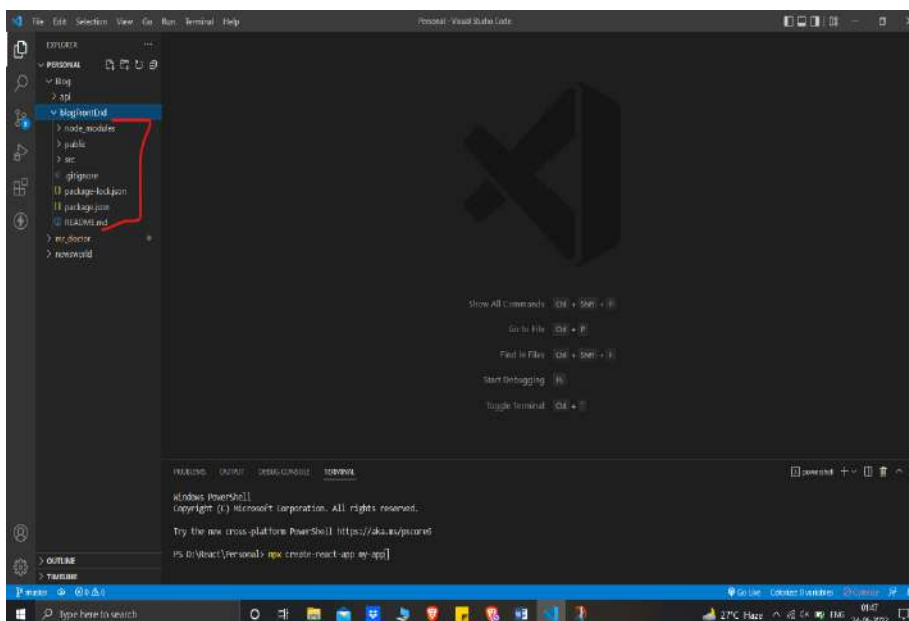
Under desired folder , open VS code and in terminal , type :

`npx create-react-app my-app`



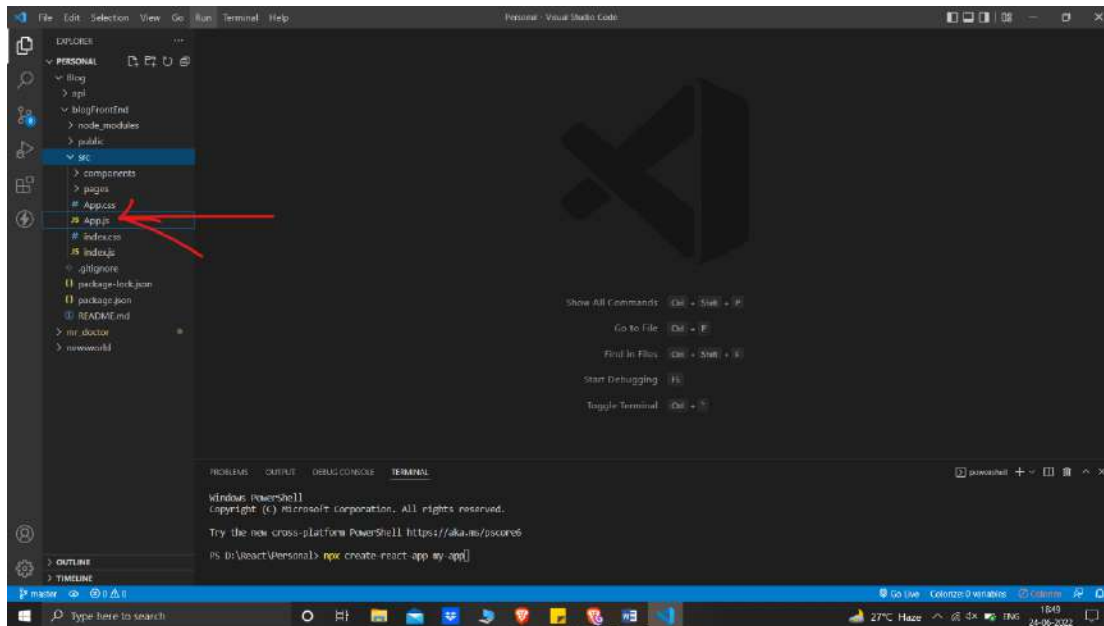
Instead of my-app you can type name of the app you want to create.

After successful creation of react app , you should be able to view a folder structure like this

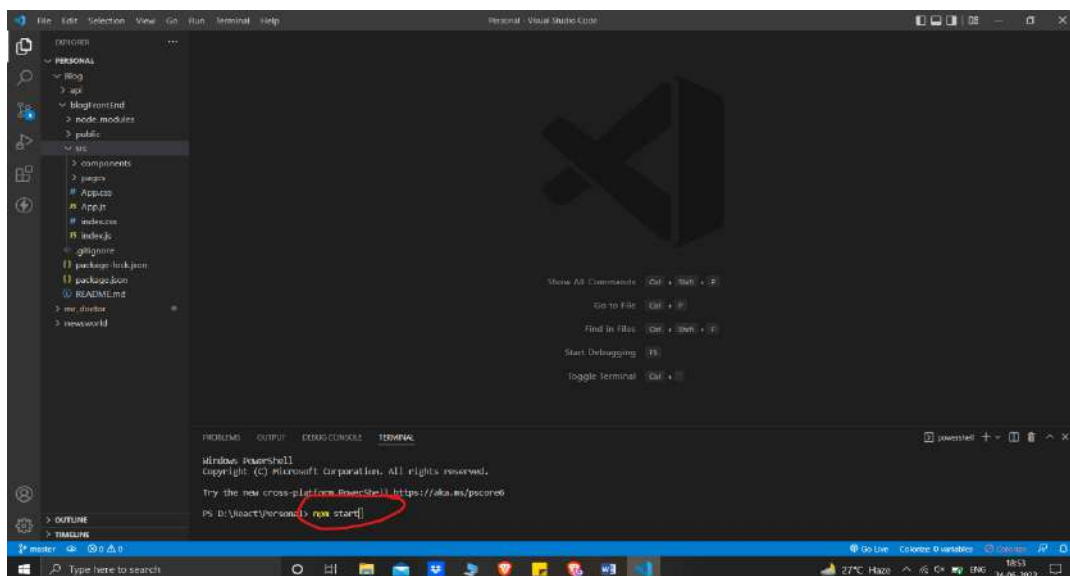


Now inside src folder of react , your will see app.js file

It is the entry point of our react application.



Now to run your react app , simply type **npm start** or **npm run start** in the terminal and hit enter :



Now wait for sometime , you will be redirected in your browser , where you can see a dummy react application.

The screenshot shows a VS Code editor window with a React application. The Explorer sidebar on the left displays a file tree with the following structure:

- PERSONAL
 - blog
 - api
 - blogFrontEnd
 - node_modules
 - public
 - src
 - components
 - pages (selected)
 - Home
 - login
 - register
 - settings
 - Single
 - Write
 - App.css
 - App.js
 - index.css
 - index.js
 - package-lock.json
 - package.json
 - README.md
 - my_doctor
 - newworld
 - OUTLINE
 - TIMELINE

The main editor displays the 'Blog.js' file, which contains the following code:

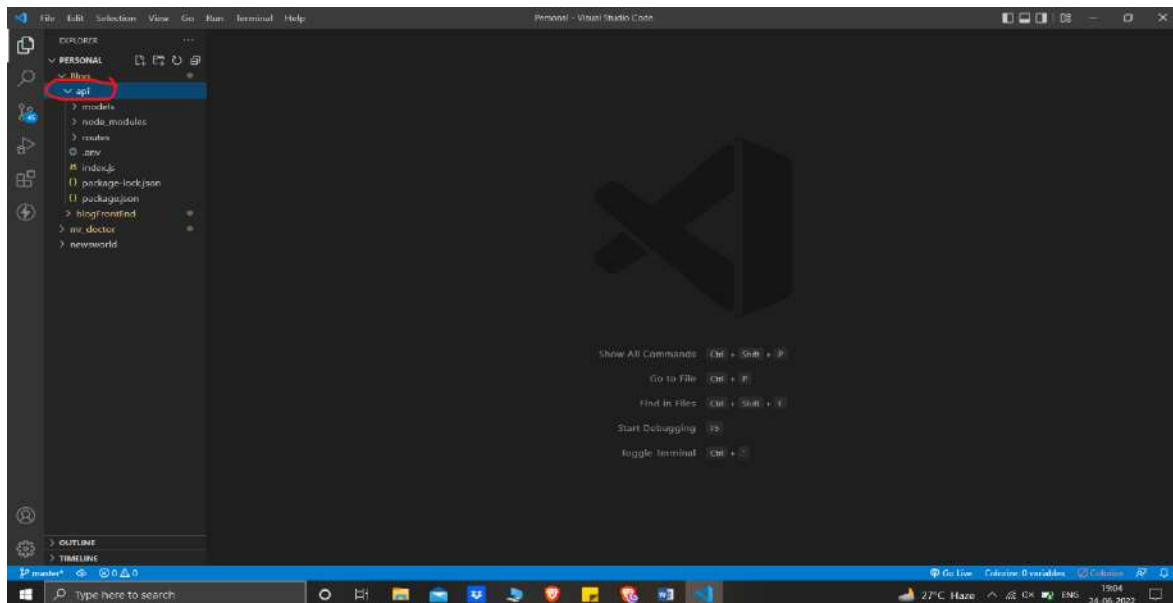
```

1 import './Navbar.css';
2 import { Link } from 'react-router-dom';
3
4 const Navbar = () => {
5   const user = false;
6   return (
7     <div className="nav">
8       <div className="navleft">
9         <div className="navicons">
10           <div className="navicon">fab fa-facebook square</div>
11           <div className="navicon">fab fa-instagram</div>
12           <div className="navicon">fab fa-twitter</div>
13         </div>
14       </div>
15       <div className="navcenter">
16         <div className="navlist">
17           <div className="navlistitem">
18             <Link to="/" className="link">
19               Home
20             </Link>
21           </div>
22           <div className="navlistitem">
23             <Link to="/" className="link">
24               About
25             </Link>
26           </div>
27           <div className="navlistitem">
28             <Link to="/" className="link">
29               Contact
30             </Link>
31           </div>
32           <div className="navlistitem">
33             <Link to="/write" className="link">
34               Write
35             </Link>
36           </div>
37         </div>
38       </div>
39       <div className="navright">
40         <div className="navlistitem">
41           {user ? <Link to="/logout" className="link">Logout</Link> : null}
42         </div>
43       </div>
44     </div>
45   );
46 }
47
48 export default Navbar;
  
```

The status bar at the bottom shows 'Ln 3, Col 1' and 'UTF-8' encoding.

If you need reference regarding front-end , you can visit to below git repository to see code of front-end

Now For backend and database , create a separate folder like api in our case :



Here you will have to install some npm packages like : node , express , mongoose , etc. for that open terminal and run below command :

```
npm install express mongoose dotenv multer nodemon
```

Here we are installing 4 packages :

1. **Express** - Express is a node js web application framework that provides broad features for building web and mobile applications.

Official website : <https://expressjs.com>

2. **Mongoose** – It is used to connect our backend with database. Mongoose acts as a front end to MongoDB

Official website : <https://mongoosejs.com>

3. **Dotenv** - It is used to hide unnecessary info inside our main file. For eg : we can use .env file to store secret keys , ids or any other sensitive data and directly call that data from .env file , without actually revealing the data.

Official website : <https://www.npmjs.com/package/dotenv>

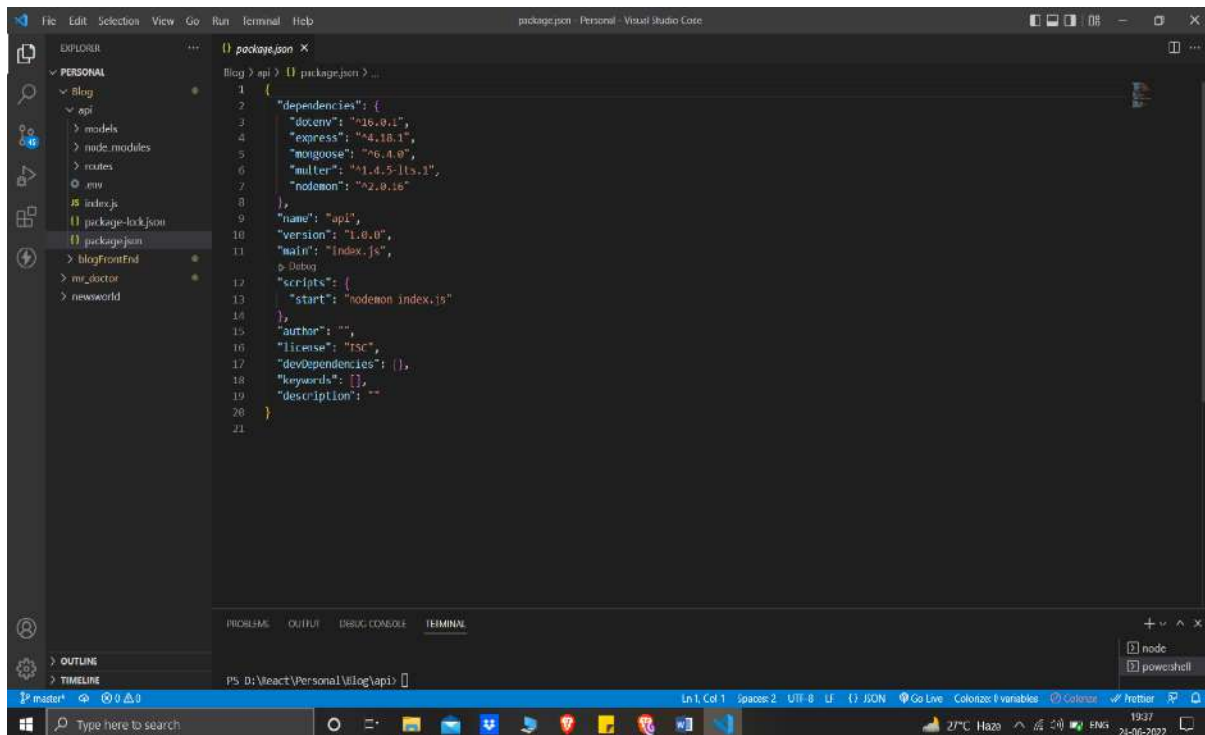
4. **Multer** – Used for uploading files.

Official website : <https://www.npmjs.com/package/multer>

5. **Nodemon** – It is a tool that helps develop Node.js based applications by automatically restarting the node application when file changes in the directory are detected

Official website : <https://nodemon.io>

Now your package.json file should look like this :



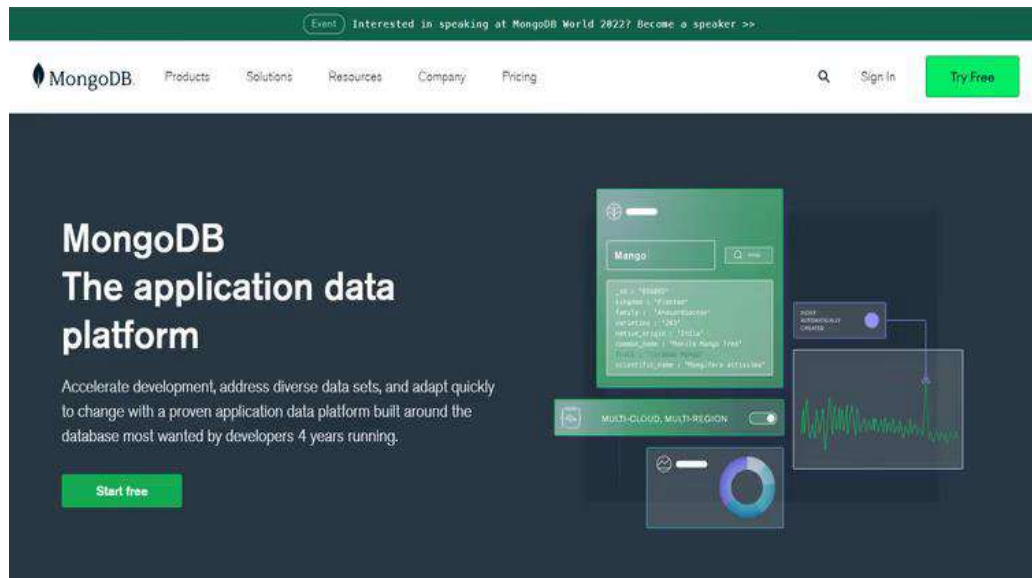
Now create a new file index.js inside api folder only .

Configuring index.js

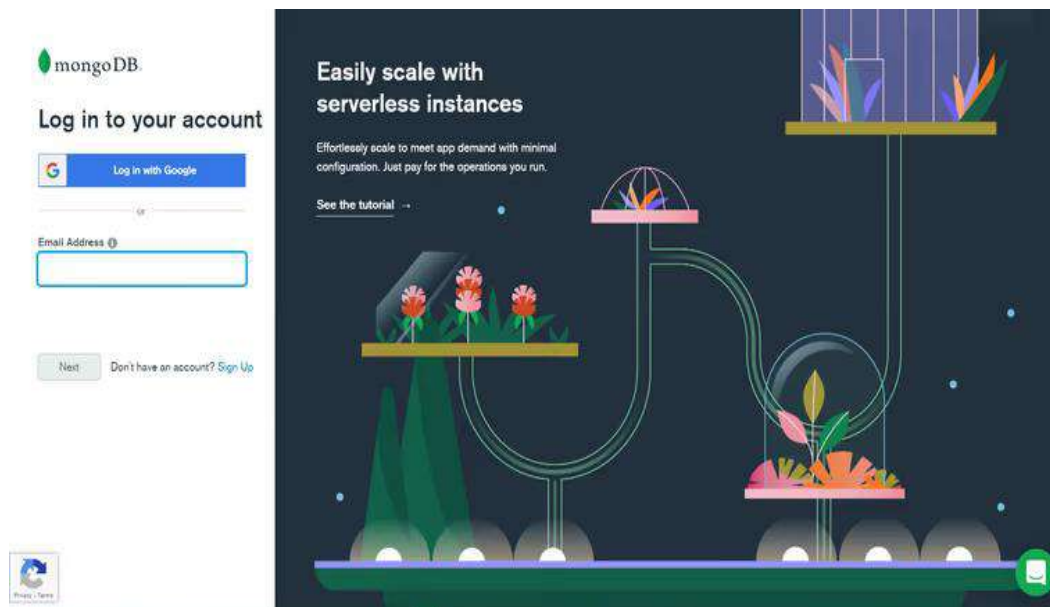
- `const express = require("express");`
- `const app = express();`
- `const dotenv = require("dotenv");`
- `const mongoose = require("mongoose");`

Now it's time to link our server application to the real database, so we'll utilize the MongoDB database, especially the MongoDB cloud Atlas version, which means we'll be hosting our database onto their cloud.

Official MongoDB website



Sign in to MongoDB



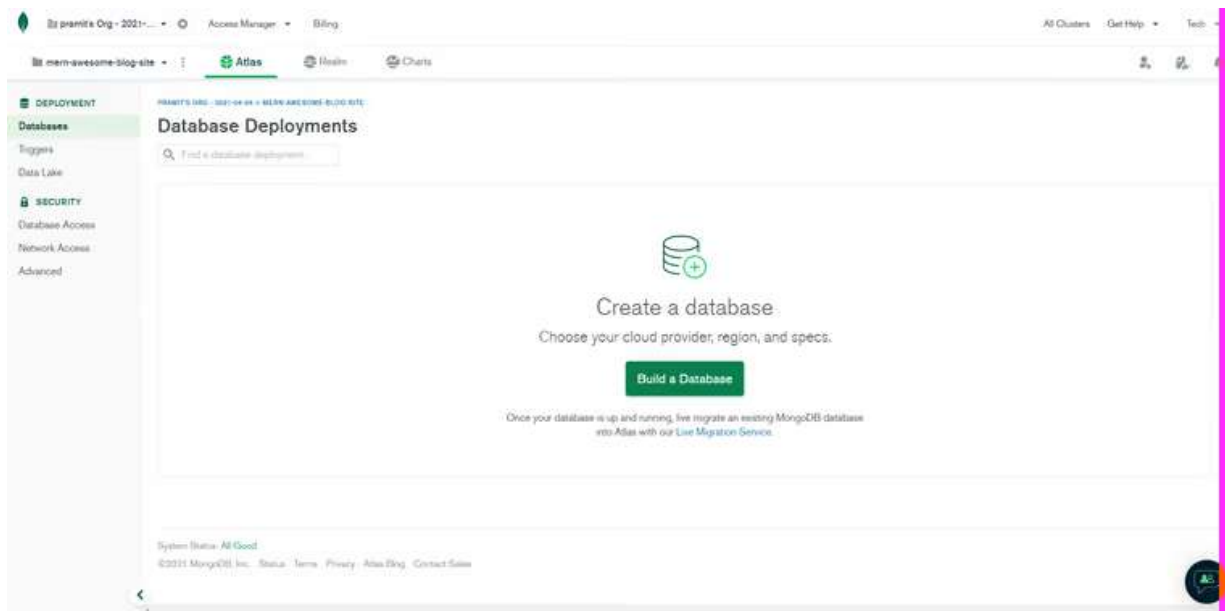
Create a Project

The screenshot shows the 'Create a Project' form in the Databricks interface. The left sidebar contains navigation links: ORGANIZATION, Projects (selected), Alerts, Activity Feed, Settings, Access Manager, Billing, Support, and Live Migration. The main content area has a breadcrumb 'PRAMI'S ORG > 2021-10-04 > PROJECTS' and a title 'Create a Project'. Below the title are two tabs: 'Name Your Project' (active) and 'Add Members'. A green 'Next' button is at the top right. The 'Name Your Project' section includes a note: 'Project names have to be unique within the organization (and other restrictions)'. A text input field contains 'mem-awesome-blog-site'. Below the field are 'Cancel' and 'Next' buttons. At the bottom, there is a system status bar: 'System Status: All Good | Last Login: 27/34/27154' and a footer: '©2021 MongoDB, Inc. | Status | Terms | Privacy | Atlas Blog | Contact Sales'.

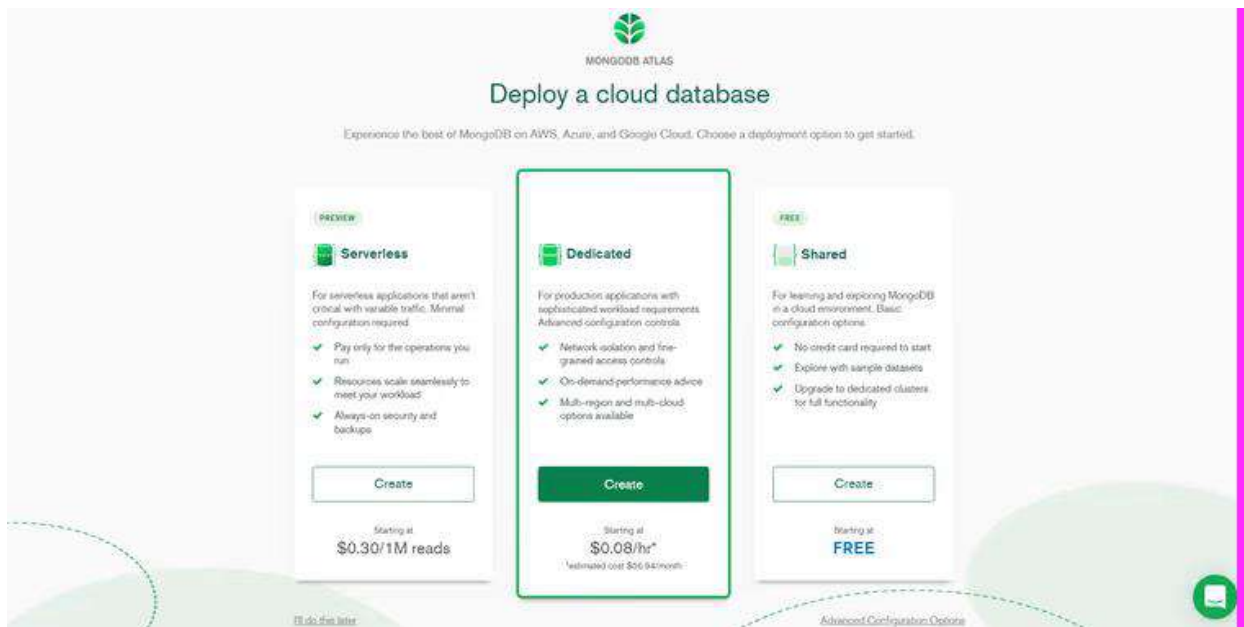
Adding members

The screenshot shows the 'Add Members and Set Permissions' form in the Databricks interface. The left sidebar is the same as the previous screenshot. The main content area has a breadcrumb 'PRAMI'S ORG > 2021-10-04 > PROJECTS' and a title 'Create a Project'. Below the title are two tabs: 'Name Your Project' (inactive) and 'Add Members' (active). A green 'Create Project' button is at the top right. The 'Add Members and Set Permissions' section includes a note: 'Invite new or existing users via email address...'. A text input field contains 'prammaratha5497@gmail.com (you)'. Below the field is a dropdown menu labeled 'Project Owner'. At the bottom are 'Cancel', 'Go Back', and 'Create Project' buttons. On the right side, there is a 'Project Member Permissions' section with a list of roles and their permissions: 'Project Owner' (Has full administration access), 'Project Cluster Manager' (Can update clusters), 'Project Data Access Admin' (Can access and modify a cluster's data and indexes, and kill operations), 'Project Data Access Read/Write' (Can access a cluster's data and indexes, and modify data), 'Project Data Access Read Only' (Can access a cluster's data and indexes), and 'Project Read Only' (May only modify personal preferences). At the bottom, there is a system status bar: 'System Status: All Good | Last Login: 27/34/27152' and a footer: '©2021 MongoDB, Inc. | Status | Terms | Privacy | Atlas Blog | Contact Sales'.

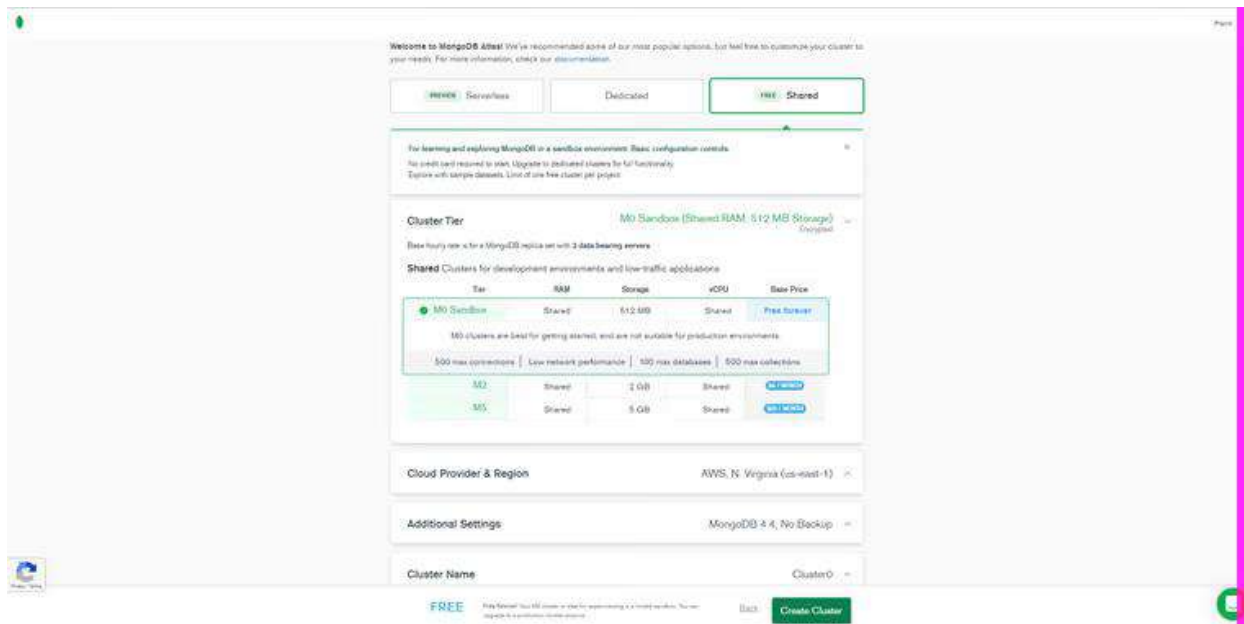
Building a database



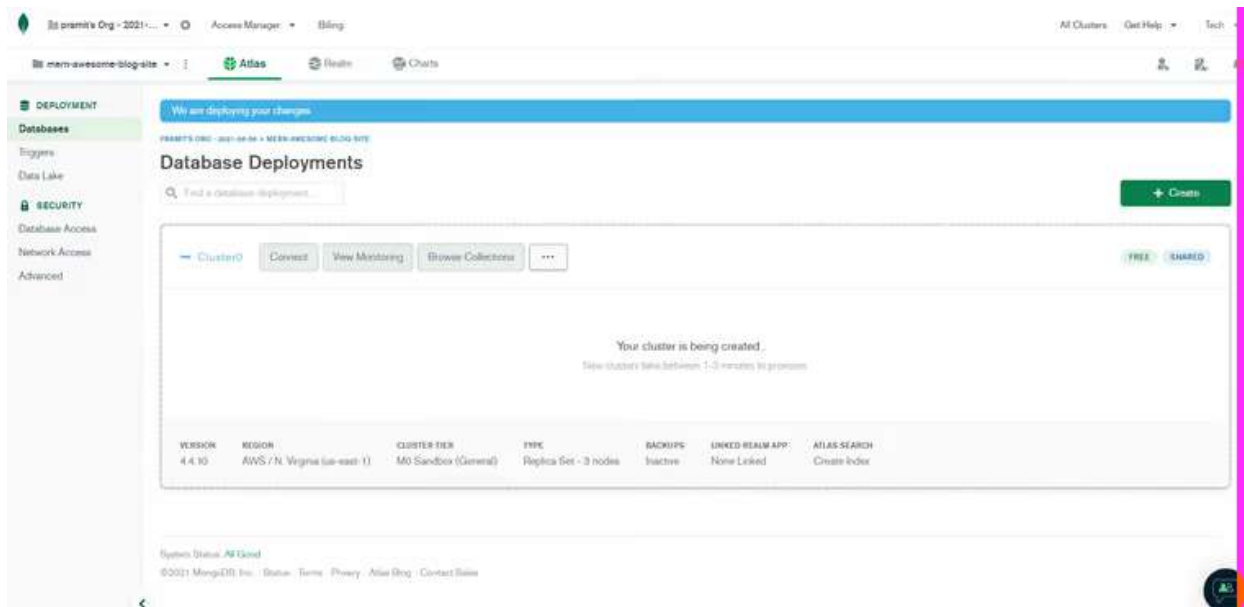
Creating a cluster



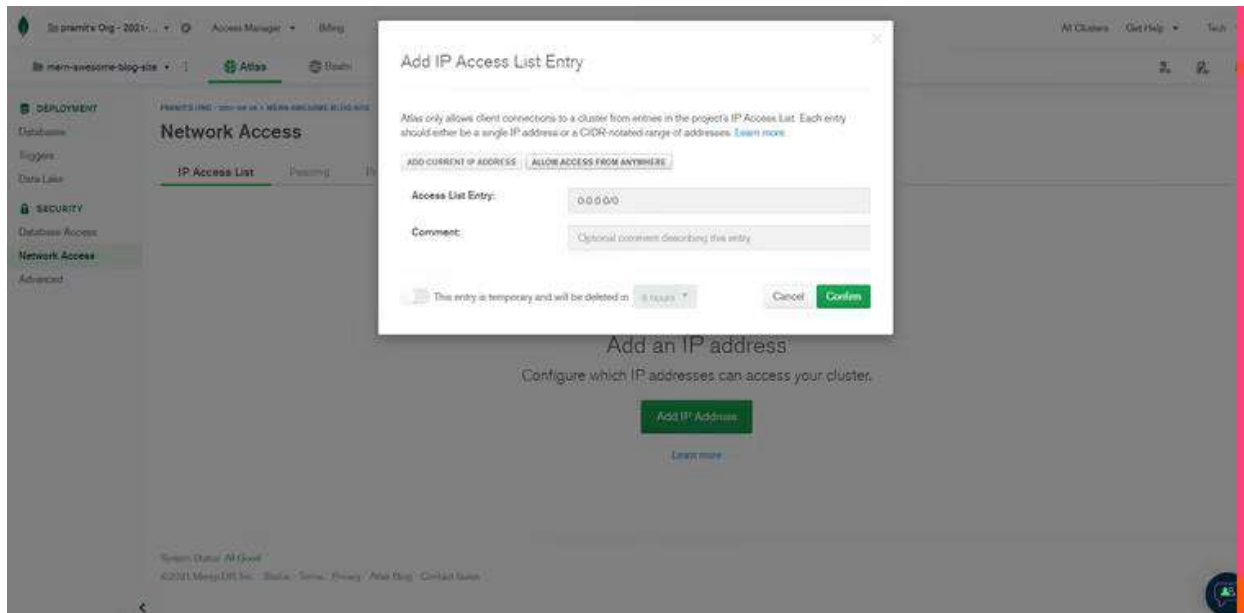
Selecting a cloud service provider



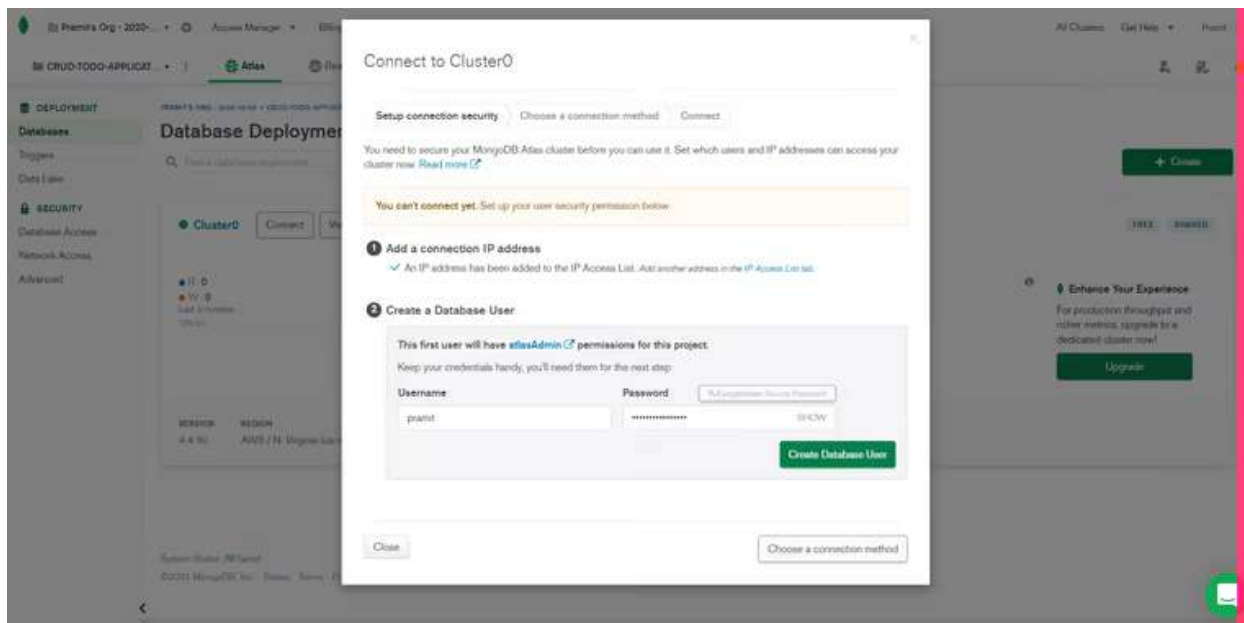
Make a cluster and wait for the cluster to be built before proceeding (usually takes around 5 -10 minutes)



Navigate to the network access tab and select "Add IP address."



In the database, create a user. You'll need the username and password for the MongoDB URI and finally, create a database user.



Now, select the Choose a connection method.

Connect to Cluster0


 Setup connection security


Choose a connection method


Connect

Choose a connection method [View documentation](#)

Get your pre-formatted connection string by selecting your tool below.

**Connect with the MongoDB Shell**
Interact with your cluster using MongoDB's interactive Javascript interface >

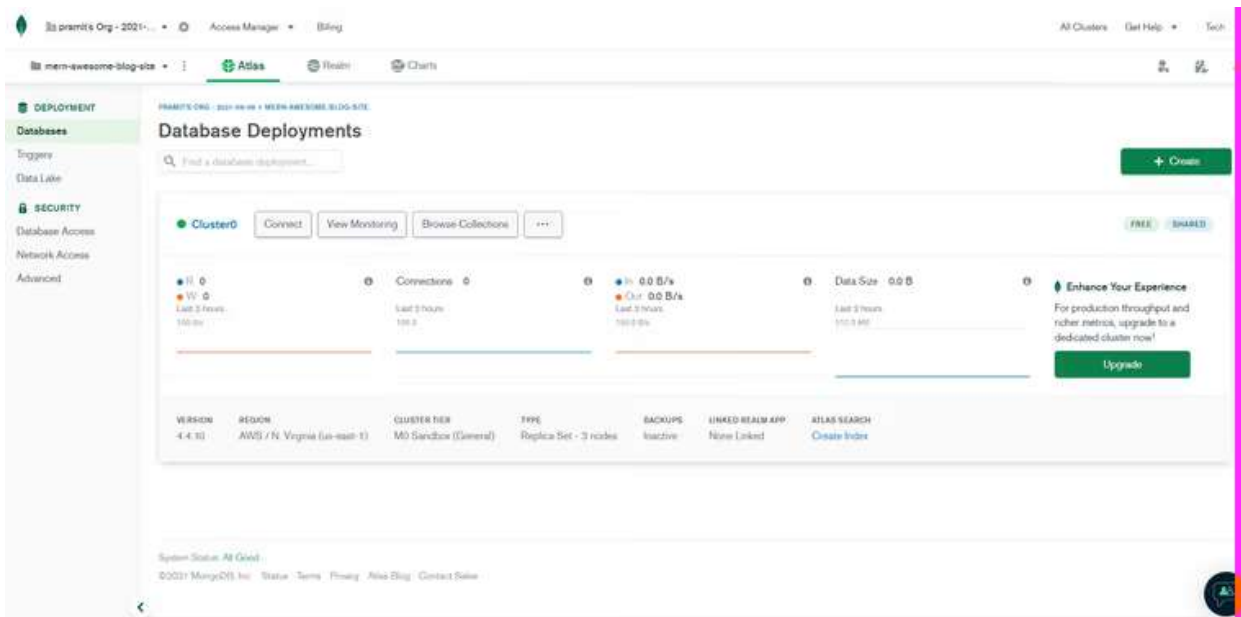
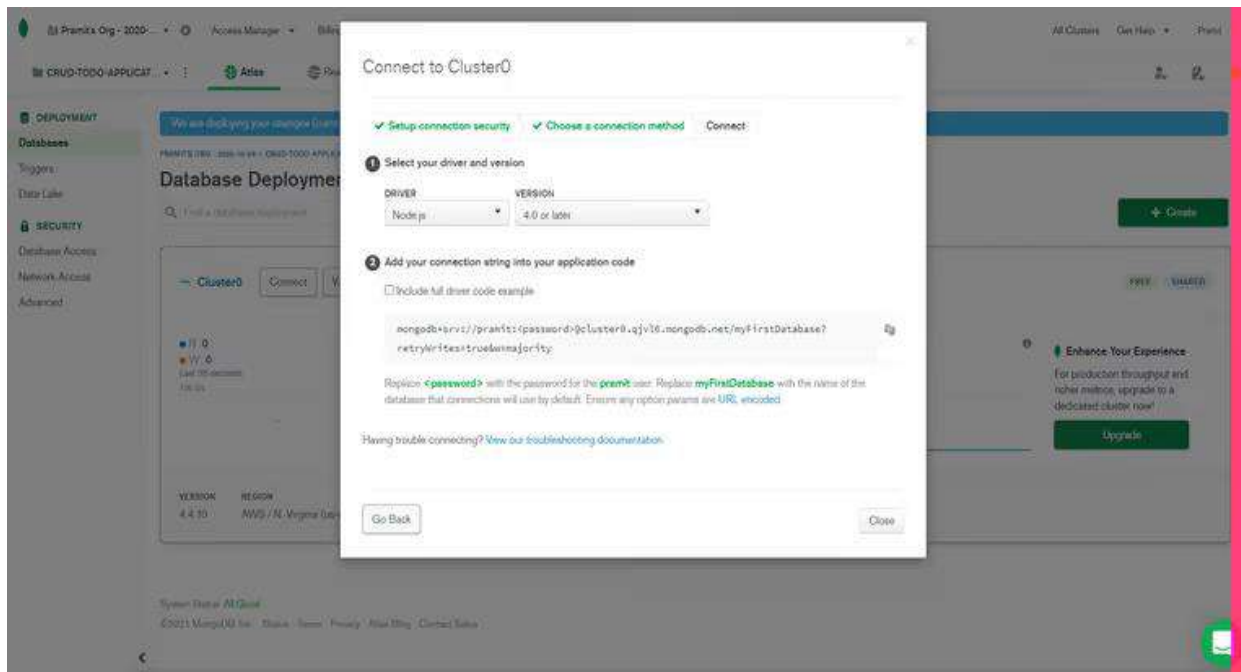
**Connect your application**
Connect your application to your cluster using MongoDB's native drivers >

**Connect using MongoDB Compass**
Explore, modify, and visualize your data with MongoDB's GUI >

Go Back

Close

Connect your application by clicking on it and finally select the correct driver and version.



Now your database is also ready to connect .

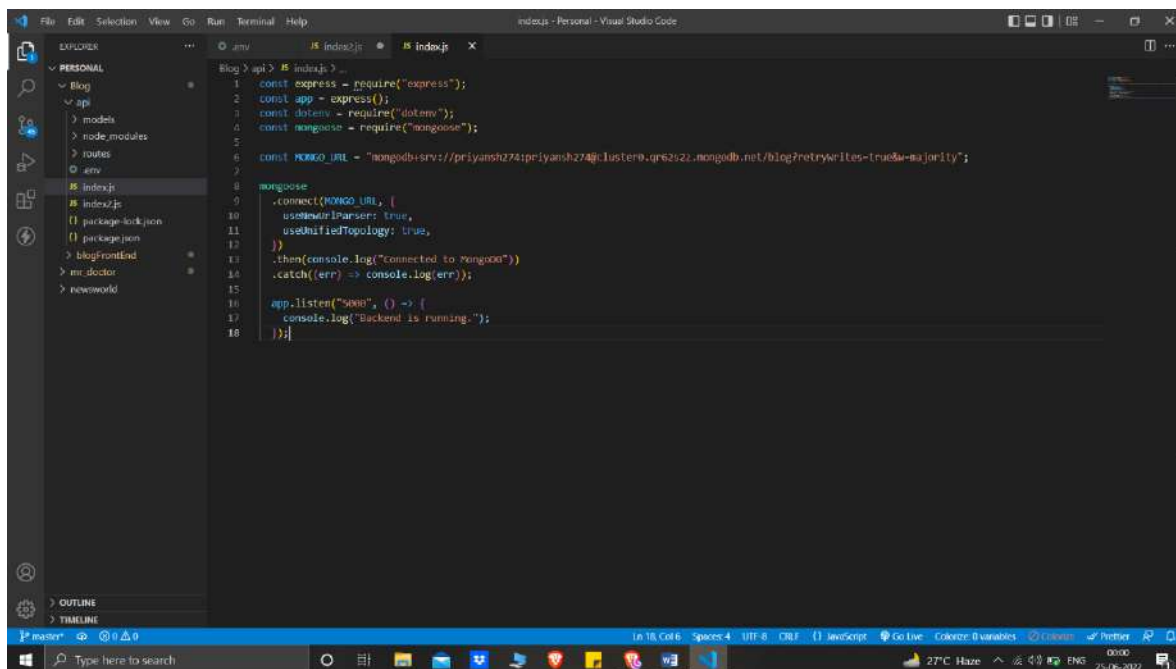
Now, inside `index.js` create a new variable and name it `MONGO_URL`. Inside it, create a string and simply paste the copied mongo DB connection URL.

Now, inside it, enter your username and password, making sure to remove all the brackets and enter your own credentials. We'll secure the credential later by creating environmental variables, but for now, let's add it this way.

The first will be the DB CONNECTION, and the second will be an object with two different options. The first is `useNewUrlParser`, which we will set to `true`, and the second is `useUnifiedTopology`, which we will also set to `true`. These objects are not required, but we will see some errors or warnings on our console.

Following that, let's chain `then()` and `catch()` because this will return a promise, so inside `.then()` will console log **“Connected to mongo”** that will be executed if our application is successfully connected and finally, if the connection to the database is not successful we will simply console log our error message.

At last, call the app and invoke `listen`, which has two parameters, the first of which is `PORT` and the second of which is the callback function that will be executed if our application is successfully connected



```
Blog > api > index.js
1  const express = require("express");
2  const app = express();
3  const dotenv = require("dotenv");
4  const mongoose = require("mongoose");
5
6  const MONGO_URL = "mongodb+srv://priyansh274ipriyansh274@cluster0.qr62t2l.mongodb.net/blog?retryWrites=true&majority";
7
8  mongoose
9    .connect(MONGO_URL, {
10      useNewUrlParser: true,
11      useUnifiedTopology: true,
12    })
13    .then(console.log("Connected to MongoDB"))
14    .catch(err => console.log(err));
15
16  app.listen("5000", () => {
17    console.log("Backend is running.");
18  });
```

That's it; we've successfully linked our server to the database.

Now you can start creating APIs and models and schemas for your blog app , according to your convenience.

If you want to refer the code, you can visit the below link:

<https://github.com/safak/youtube/tree/blog-rest-api>