**Building your own AI SaaS Chat-bot by MERN(MongoDB,Express,React and Node.js) Stack**

[[Sanikakotgire](https://medium.com/@sanikakotgire2003?source=post_page-----306a37a4845d--------------------------------)](https://medium.com/@sanikakotgire2003?source=post_page-----306a37a4845d--------------------------------)

[Sanikakotgire](https://medium.com/@sanikakotgire2003?source=post_page-----306a37a4845d--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F295c041d844e&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40sanikakotgire2003%2Fbuilding-your-own-ai-chat-bot-using-mern-mongodb-express-react-and-node-js-stack-306a37a4845d&user=Sanikakotgire&userId=295c041d844e&source=post_page-295c041d844e----306a37a4845d---------------------post_header-----------)

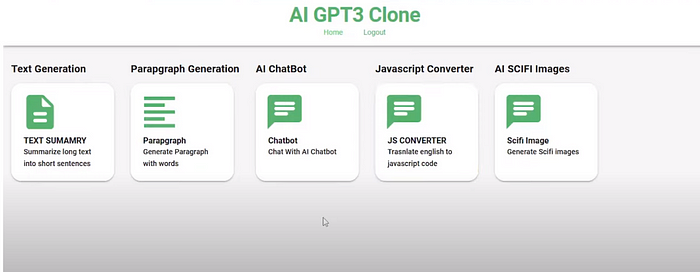
9 min read

·

Nov 14

109

In this project, we will delve into the step-by-step process of constructing a Chat-bot AI using the MERN Stack. From setting up the backend server to designing an interactive and dynamic front end, we will explore the intricacies of integrating AI capabilities to create a responsive and intelligent conversational interface.



**Interface of the designed Chatbot**

**STEP 1 : Setup your pc with all the required stack:**

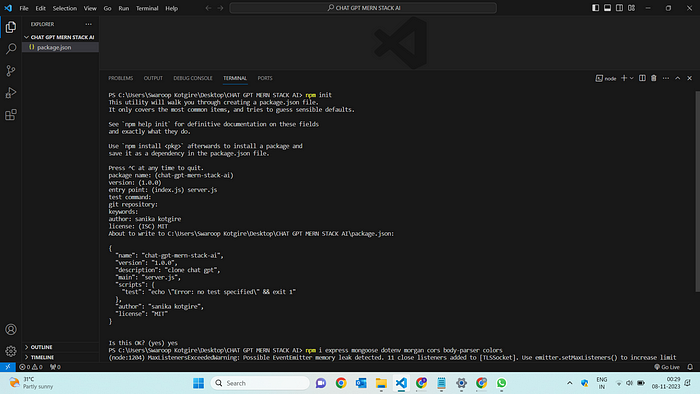
*1)Vs-Code Editor*

*2)Mongo DB Compass*

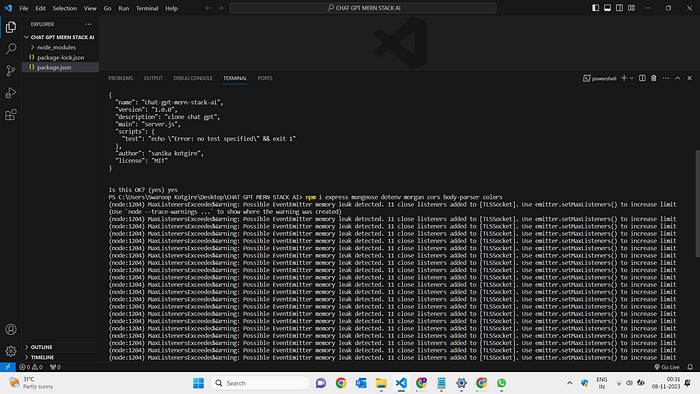
*3)Node.js*

*4)Express(npm i express)*

*5)package. json file(npm init)*

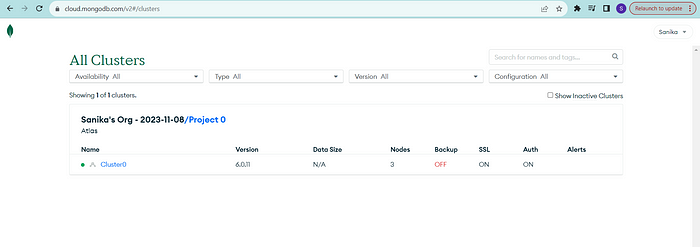


*6)Node Modules (npm i express mongoose dotenv morgan cors body-parser colors)*

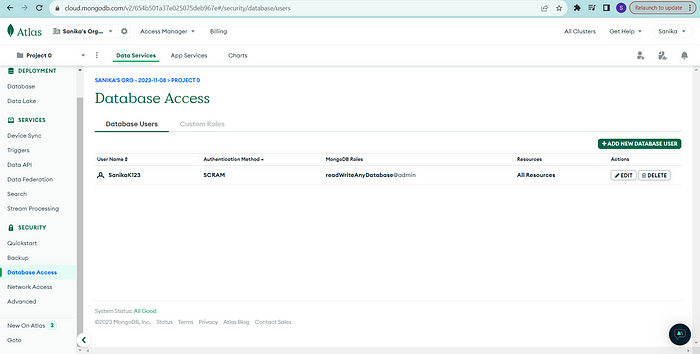


**STEP 2:Mongodb Connection and React setup**

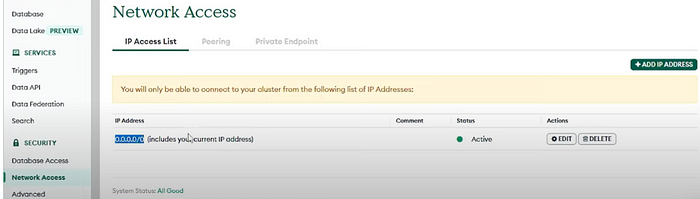
1. Create a Free Cluster using any Cloud Provider (AWS, Azure or GCP) using M0 Sandbox(Shared RAM,512 MB storage)



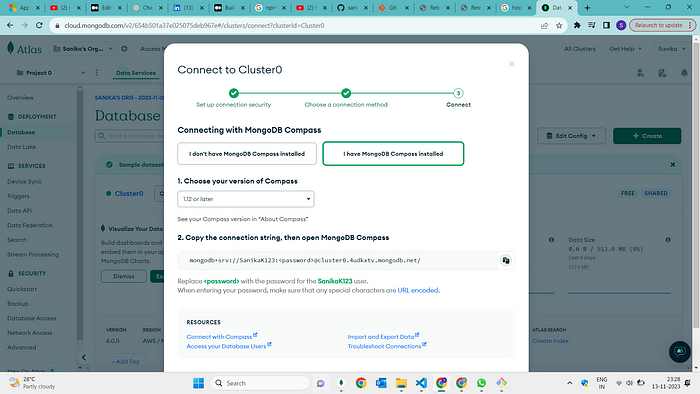
2. Create a User



3.Create a network access with the same IP



4.Connect with your Mongodb compass ,copy the provided URI.

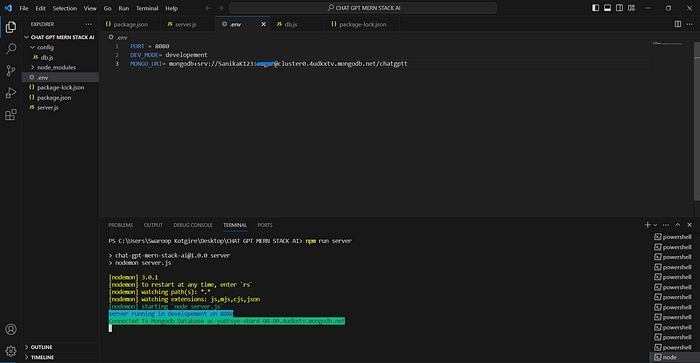


5.Make a new file *‘db.js’* and enter the below code for connection.

const mongoose = require("mongoose");  
const colors = require("colors");  
  
const connectDB = async () => {  
 try {  
 await mongoose.connect(process.env.MONGO\_URI);  
 console.log(  
 `Connected To Mongodb Database ${mongoose.connection.host}`.bgGreen.white  
 );  
 } catch (error) {  
 console.log(`Mognodb Database Error ${error}`.bgRed.white);  
 }  
};  
  
module.exports = connectDB;

6.Now paste it in .env file for the MONGO\_URI to connect cluster with your data base. Also replace the password with your actual password and then run the command provided below:

npm run server

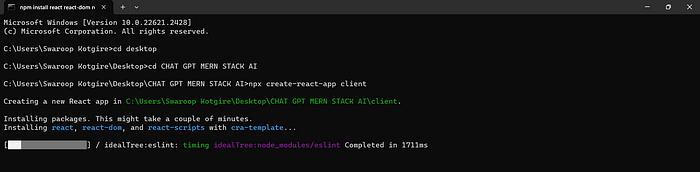


We have now successfully connected our database cluster.

7.Run the commands below for React Setups and other packages.

npx create-react-app-client  
npm i axios react-router-dom  
npm install @mui/material @emotion/react @emotion/styled  
npm install @mui/icons-material

npm install axios react-router-dom @mui/material @emotion/react @emotion/styled @mui/icons-material: Install the required dependencies. Axios is for making HTTP requests, React Router DOM is for routing in React, and @mui/material and related packages are for Material-UI components.



**STEP 3:User Model hashing password and token cookie generation:**

1. Run the command for bcrypt setup and cookie generation. This command uses npm install to install the specified packages: bcryptjs for password hashing, jsonwebtoken for handling JSON Web Tokens, and cookie for working with cookies in a Node.js environment.

npm i bcryptjs jsonwebtoken cookie

2.Create a file under Models as ‘*userModel.js’* and enter the code provided:

const mongoose = require("mongoose");  
const bcrypt = require("bcryptjs");  
const JWT = require("jsonwebtoken");  
const cookie = require("cookie");  
  
//models  
const userSchema = new mongoose.Schema({  
 username: {  
 type: String,  
 required: [true, "USername is Required"],  
 },  
 email: {  
 type: String,  
 required: [true, "Email is required"],  
 unique: true,  
 },  
 password: {  
 type: String,  
 required: [true, "Password is required"],  
 minlength: [6, "Password length should be 6 character long"],  
 },  
 customerId: {  
 type: String,  
 default: "",  
 },  
 subscription: {  
 type: String,  
 default: "",  
 },  
});  
  
//hashed password  
userSchema.pre("save", async function (next) {  
 //update  
 if (!this.isModified("password")) {  
 next();  
 }  
 const salt = await bcrypt.genSalt(10);  
 this.password = await bcrypt.hash(this.password, salt);  
 next();  
});  
  
//match password  
userSchema.methods.matchPassword = async function (password) {  
 return await bcrypt.compare(password, this.password);  
};  
  
//SIGN TOKEN  
userSchema.methods.getSignedToken = function (res) {  
 const acccesToken = JWT.sign(  
 { id: this.\_id },  
 process.env.JWT\_ACCESS\_SECRET,  
 { expiresIn: process.env.JWT\_ACCESS\_EXPIREIN }  
 );  
 const refreshToken = JWT.sign(  
 { id: this.\_id },  
 process.env.JWT\_REFRESH\_TOKEN,  
 { expiresIn: process.env.JWT\_REFRESH\_EXIPREIN }  
 );  
 res.cookie("refreshToken", `${refreshToken}`, {  
 maxAge: 86400 \* 7000,  
 httpOnly: true,  
 });  
};  
  
const User = mongoose.model("User", userSchema);  
  
module.exports = User;

3.Add this to your ‘*.env’*file for cookie generation

JWT\_ACCESS\_SECRET=HHZHSJCKJSKXLSZKXDSK23sgfg  
JWT\_ACCESS\_EXPIREIN= 15min  
JWT\_REFRESH\_TOKEN=AFGBDNSKKSKSABHJ667agghh  
JWT\_REFRESH\_EXPIREIN=15day

4.Create another file under Routes as *‘authRoutes.js’* for routing password and enter the code below:

const express = require("express");  
const {  
 registerContoller,  
 loginController,  
 logoutController,  
} = require("../controllers/authController");  
  
//router object  
const router = express.Router();  
  
//routes  
// REGISTER  
router.post("/register", registerContoller);  
  
//LOGIN  
router.post("/login", loginController);  
  
//LOGOUT  
router.post("/logout", logoutController);  
  
module.exports = router;

**STEP 4: Setting up Middlewares and login register API:**

1. Setup a file for error generation and detection under utils as *‘errroResponse.js’*and enter the code provided below.

class errorResponse extends Error {  
 constructor(message, statusCode) {  
 super(message);  
 this.statusCode = statusCode;  
 }  
 }  
   
 module.exports = errorResponse;

2.Update the*‘server.js’* file for server connection to your database.

const express = require("express");  
const morgan = require("morgan");  
const cors = require("cors");  
const bodyParser = require("body-parser");  
const colors = require("colors");  
const dotenv = require("dotenv");  
const connectDB = require("./config/db");  
const errorHandler = require("./middelwares/errorMiddleware");  
  
//routes path  
const authRoutes = require("./routes/authRoutes");  
  
//dotenv  
dotenv.config();  
  
//mongo connection  
connectDB();  
  
//rest object  
const app = express();  
  
//middlewares  
app.use(cors());  
app.use(express.json());  
app.use(bodyParser.urlencoded({ extended: true }));  
app.use(morgan("dev"));  
app.use(errorHandler);  
  
const PORT = process.env.PORT || 8080;  
  
//API routes  
app.use("/api/v1/auth", authRoutes);  
app.use("/api/v1/openai", require("./routes/openaiRoutes"));  
  
//listen server  
app.listen(PORT, () => {  
 console.log(  
 `Server Running in ${process.env.DEV\_MODE} mode on port no ${PORT}`.bgCyan  
 .white  
 );  
});

3.Create another file under middelwares as *‘errormiddleware.js’*and enter the code provided.

const errorResponse = require("../utils/errroResponse");  
  
const errorHandler = (err, req, res, next) => {  
 let error = { ...err };  
 error.message = err.message;  
  
 //mongoose cast Error  
 if (err.name === "castError") {  
 const message = "Resources Not Found";  
 error = new errorResponse(message, 404);  
 }  
 //duplicate key error  
 if (err.code === 11000) {  
 const message = "Duplicate field value enterd";  
 error = new errorResponse(message, 400);  
 }  
 //mongoose validation  
 if (err.name === "ValidationError") {  
 const messgae = Object.values(err.errors).map((val) => val.message);  
 error = new errorResponse(message, 400);  
 res.status(error.statusCode || 500).json({  
 success: false,  
 error: error.message || "Server Error",  
 });  
 }  
};  
  
module.exports = errorHandler;

**STEP 5: Navbar and Routing**

1. To proceed with our homepage lets enter the code under the pages to the file *‘Homepage.js’*

import React from "react";  
import { Box, Typography, Card, Stack } from "@mui/material";  
import { useNavigate } from "react-router-dom";  
import DescriptionRounded from "@mui/icons-material/DescriptionRounded";  
import FormatAlignLeftOutlined from "@mui/icons-material/FormatAlignLeftOutlined";  
import ChatRounded from "@mui/icons-material/ChatRounded";  
const Homepage = () => {  
 const navigate = useNavigate();  
 return (  
 <>  
 <Box sx={{ display: "flex", flexDirection: "row" }}>  
 <Box p={2}>  
 <Typography variant="h4" mb={2} fontWeight="bold">  
 Text Generation  
 </Typography>  
 <Card  
 onClick={() => navigate("/summary")}  
 sx={{  
 boxShadow: 2,  
 borderRadius: 5,  
 height: 190,  
 width: 200,  
 "&:hover": {  
 border: 2,  
 boxShadow: 0,  
 borderColor: "primary.dark",  
 cursor: "pointer",  
 },  
 }}  
 >  
 <DescriptionRounded  
 sx={{ fontSize: 80, color: "primary.main", mt: 2, ml: 2 }}  
 />  
 <Stack p={3} pt={0}>  
 <Typography fontWeight="bold" variant="h5">  
 TEXT SUMAMRY  
 </Typography>  
 <Typography variant="h6">  
 Summarize long text into short sentences  
 </Typography>  
 </Stack>  
 </Card>  
 </Box>  
 <Box p={2}>  
 <Typography variant="h4" mb={2} fontWeight="bold">  
 Parapgraph Generation  
 </Typography>  
 <Card  
 onClick={() => navigate("/paragraph")}  
 sx={{  
 boxShadow: 2,  
 borderRadius: 5,  
 height: 190,  
 width: 200,  
 "&:hover": {  
 border: 2,  
 boxShadow: 0,  
 borderColor: "primary.dark",  
 cursor: "pointer",  
 },  
 }}  
 >  
 <FormatAlignLeftOutlined  
 sx={{ fontSize: 80, color: "primary.main", mt: 2, ml: 2 }}  
 />  
 <Stack p={3} pt={0}>  
 <Typography fontWeight="bold" variant="h5">  
 Parapgraph  
 </Typography>  
 <Typography variant="h6">  
 Generate Paragraph with words  
 </Typography>  
 </Stack>  
 </Card>  
 </Box>  
 <Box p={2}>  
 <Typography variant="h4" mb={2} fontWeight="bold">  
 AI ChatBot  
 </Typography>  
 <Card  
 onClick={() => navigate("/chatbot")}  
 sx={{  
 boxShadow: 2,  
 borderRadius: 5,  
 height: 190,  
 width: 200,  
 "&:hover": {  
 border: 2,  
 boxShadow: 0,  
 borderColor: "primary.dark",  
 cursor: "pointer",  
 },  
 }}  
 >  
 <ChatRounded  
 sx={{ fontSize: 80, color: "primary.main", mt: 2, ml: 2 }}  
 />  
 <Stack p={3} pt={0}>  
 <Typography fontWeight="bold" variant="h5">  
 Chatbot  
 </Typography>  
 <Typography variant="h6">Chat With AI Chatbot</Typography>  
 </Stack>  
 </Card>  
 </Box>  
 <Box p={2}>  
 <Typography variant="h4" mb={2} fontWeight="bold">  
 Javascript Converter  
 </Typography>  
 <Card  
 onClick={() => navigate("/js-converter")}  
 sx={{  
 boxShadow: 2,  
 borderRadius: 5,  
 height: 190,  
 width: 200,  
 "&:hover": {  
 border: 2,  
 boxShadow: 0,  
 borderColor: "primary.dark",  
 cursor: "pointer",  
 },  
 }}  
 >  
 <ChatRounded  
 sx={{ fontSize: 80, color: "primary.main", mt: 2, ml: 2 }}  
 />  
 <Stack p={3} pt={0}>  
 <Typography fontWeight="bold" variant="h5">  
 JS CONVERTER  
 </Typography>  
 <Typography variant="h6">  
 Trasnlate english to javascript code  
 </Typography>  
 </Stack>  
 </Card>  
 </Box>  
 <Box p={2}>  
 <Typography variant="h4" mb={2} fontWeight="bold">  
 AI SCIFI Images  
 </Typography>  
 <Card  
 onClick={() => navigate("/scifi-image")}  
 sx={{  
 boxShadow: 2,  
 borderRadius: 5,  
 height: 190,  
 width: 200,  
 "&:hover": {  
 border: 2,  
 boxShadow: 0,  
 borderColor: "primary.dark",  
 cursor: "pointer",  
 },  
 }}  
 >  
 <ChatRounded  
 sx={{ fontSize: 80, color: "primary.main", mt: 2, ml: 2 }}  
 />  
 <Stack p={3} pt={0}>  
 <Typography fontWeight="bold" variant="h5">  
 Scifi Image  
 </Typography>  
 <Typography variant="h6">Generate Scifi images</Typography>  
 </Stack>  
 </Card>  
 </Box>  
 </Box>  
 </>  
 );  
};  
  
export default Homepage;

2.Now run the concurrently command as the concurrently package provides an easy way to run multiple commands in parallel from the command line. This package allows you to specify a list of commands to run, along with any arguments or options, and then executes them all at the same time.

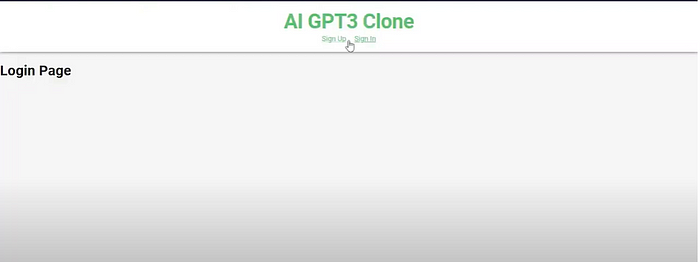
npm i concurrently

3.Use the code provided under Navbar for *‘Navbar.js’*

import React from "react";  
import { Box, Typography, useTheme } from "@mui/material";  
import { useNavigate } from "react-router-dom";  
import { NavLink } from "react-router-dom";  
import axios from "axios";  
import toast from "react-hot-toast";  
const Navbar = () => {  
 const theme = useTheme();  
 const navigate = useNavigate();  
 const loggedIn = JSON.parse(localStorage.getItem("authToken"));  
  
 //handle logout  
 const handleLogout = async () => {  
 try {  
 await axios.post("/api/v1/auth/logout");  
 localStorage.removeItem("authToken");  
 toast.success("logout successfully ");  
 navigate("/login");  
 } catch (error) {  
 console.log(error);  
 }  
 };  
 return (  
 <Box  
 width={"100%"}  
 backgroundColor={theme.palette.background.alt}  
 p="1rem 6%"  
 textAlign={"center"}  
 sx={{ boxShadow: 3, mb: 2 }}  
 >  
 <Typography variant="h1" color="primary" fontWeight="bold">  
 AI GPT3 Clone  
 </Typography>  
 {loggedIn ? (  
 <>  
 <NavLink to="/" p={1}>  
 Home  
 </NavLink>  
 <NavLink to="/login" onClick={handleLogout} p={1}>  
 Logout  
 </NavLink>  
 </>  
 ) : (  
 <>  
 <NavLink to="/register" p={1}>  
 Sign Up  
 </NavLink>  
 <NavLink to="/login" p={1}>  
 Sign In  
 </NavLink>  
 </>  
 )}  
 </Box>  
 );  
};  
  
export default Navbar;

4.Use the given command to obtain the login and register page.

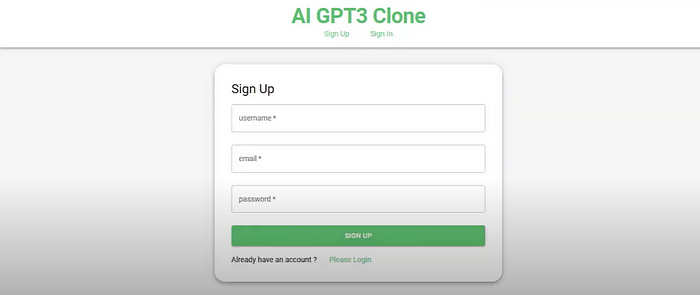
npm run dev



5.Enter the code for *‘Register.js’* for user password authentication.

import React, { useState } from "react";  
import { Link, useNavigate } from "react-router-dom";  
import toast from "react-hot-toast";  
import axios from "axios";  
import {  
 Box,  
 Typography,  
 useTheme,  
 useMediaQuery,  
 TextField,  
 Button,  
 Alert,  
 Collapse,  
} from "@mui/material";  
  
const Register = () => {  
 const theme = useTheme();  
 const navigate = useNavigate();  
 //media  
 const isNotMobile = useMediaQuery("(min-width: 1000px)");  
 // states  
 const [username, setUsername] = useState("");  
 const [email, setEmail] = useState("");  
 const [password, setPassword] = useState("");  
 const [error, setError] = useState("");  
  
 //register ctrl  
 const handleSubmit = async (e) => {  
 e.preventDefault();  
 try {  
 await axios.post("/api/v1/auth/register", { username, email, password });  
 toast.success("User Register Successfully");  
 navigate("/login");  
 } catch (err) {  
 console.log(error);  
 if (err.response.data.error) {  
 setError(err.response.data.error);  
 } else if (err.message) {  
 setError(err.message);  
 }  
 setTimeout(() => {  
 setError("");  
 }, 5000);  
 }  
 };  
 return (  
 <Box  
 width={isNotMobile ? "40%" : "80%"}  
 p={"2rem"}  
 m={"2rem auto"}  
 borderRadius={5}  
 sx={{ boxShadow: 5 }}  
 backgroundColor={theme.palette.background.alt}  
 >  
 <Collapse in={error}>  
 <Alert severity="error" sx={{ mb: 2 }}>  
 {error}  
 </Alert>  
 </Collapse>  
 <form onSubmit={handleSubmit}>  
 <Typography variant="h3">Sign Up</Typography>  
 <TextField  
 label="username"  
 required  
 margin="normal"  
 fullWidth  
 value={username}  
 onChange={(e) => {  
 setUsername(e.target.value);  
 }}  
 />  
 <TextField  
 label="email"  
 type="email"  
 required  
 margin="normal"  
 fullWidth  
 value={email}  
 onChange={(e) => {  
 setEmail(e.target.value);  
 }}  
 />  
 <TextField  
 label="password"  
 type="password"  
 required  
 margin="normal"  
 fullWidth  
 value={password}  
 onChange={(e) => {  
 setPassword(e.target.value);  
 }}  
 />  
 <Button  
 type="submit"  
 fullWidth  
 variant="contained"  
 size="large"  
 sx={{ color: "white", mt: 2 }}  
 >  
 Sign Up  
 </Button>  
 <Typography mt={2}>  
 Already have an account ? <Link to="/login">Please Login</Link>  
 </Typography>  
 </form>  
 </Box>  
 );  
};  
  
export default Register;

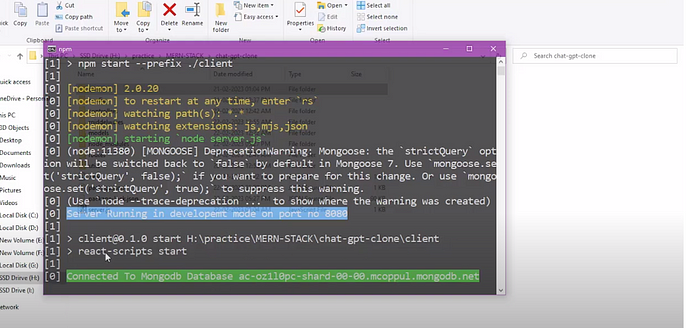
Now run the same command and to get the authentication setup ready.



6.Run the command on your terminal package for displaying toasts in a React application

npm i react-hot-toast

7.Now for the password credentials run the command on your CLI to ensure whether it is still connected to your Mongodb database.



The green visibility confirms the connection.

**STEP 6: Routing all the pages and interfaces**

1. Make a file under routes as *‘openaiRoutes.js’ .* This file will link all the pages and directories under the AI routes an provide the desired functionality.

const express = require("express");  
const {  
 summaryController,  
 paragraphController,  
 chatbotController,  
 jsconverterController,  
 scifiImageController,  
} = require("../controllers/openiaController");  
  
const router = express.Router();  
  
//route  
router.post("/summary", summaryController);  
router.post("/paragraph", paragraphController);  
router.post("/chatbot", chatbotController);  
router.post("/js-converter", jsconverterController);  
router.post("/scifi-image", scifiImageController);  
  
module.exports = router;

2.Now under the Pages make the following files for pages:

* *Homepage.js*
* *Chatbot.js*
* *Summary.js*
* *Paragraph.js*
* *Register.js*
* *JsConverter.js*
* *ScifiImage.js*

Now use this**Git hub repository** to enter the code for all the desired pages:

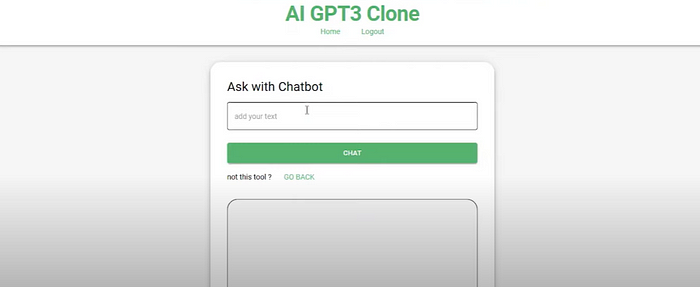
**[project1/client/src/pages at main · sanika6969/project1](https://github.com/sanika6969/project1/tree/main/client/src/pages?source=post_page-----306a37a4845d--------------------------------" \t "_blank)**

[Contribute to sanika6969/project1 development by creating an account on GitHub.](https://github.com/sanika6969/project1/tree/main/client/src/pages?source=post_page-----306a37a4845d--------------------------------" \t "_blank)

[github.com](https://github.com/sanika6969/project1/tree/main/client/src/pages?source=post_page-----306a37a4845d--------------------------------" \t "_blank)

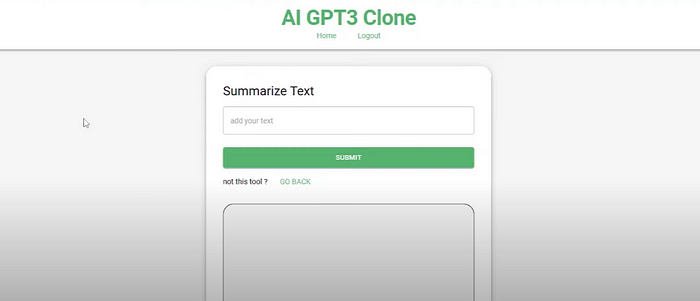
After entering the provided code, the page wise output can be obtained as follows:

* *Chatbot.js*



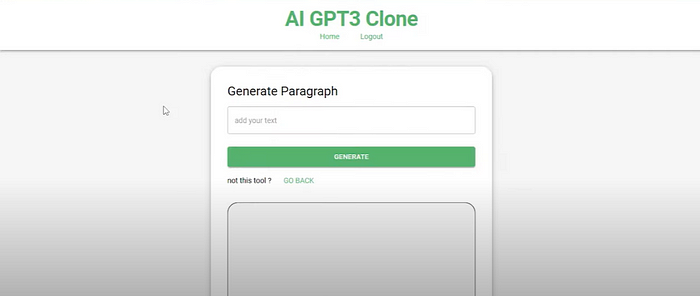
**PAGE 1**

* *Summary.js*



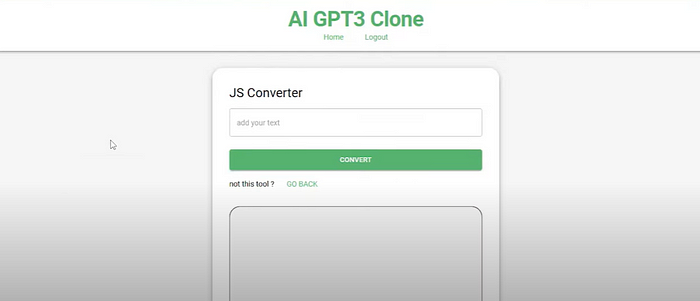
**PAGE 2**

* *Paragraph.js*



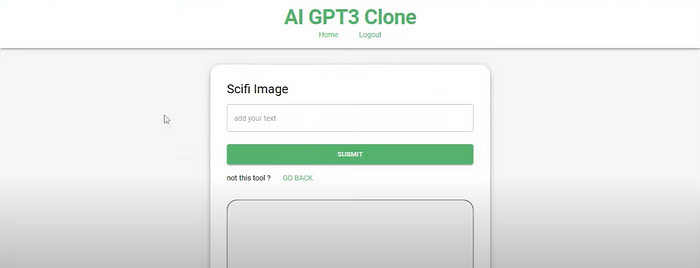
**PAGE 3**

* *JsConverter.js*



**PAGE 4**

* *ScifiImage.js*



**PAGE 5**

We’ve now completed our project .

Refer [**https://github.com/sanika6969/project1/tree/main**](https://github.com/sanika6969/project1/tree/main) for other codes and files.

Happy learning:)

#GDG#GDGpune#blogathonS02

Thank you!