

MERN Workshop

MERN – Mongo Express React Node



GURU NANAK INSTITUTE OF MANAGEMENT
AND TECHNOLOGY

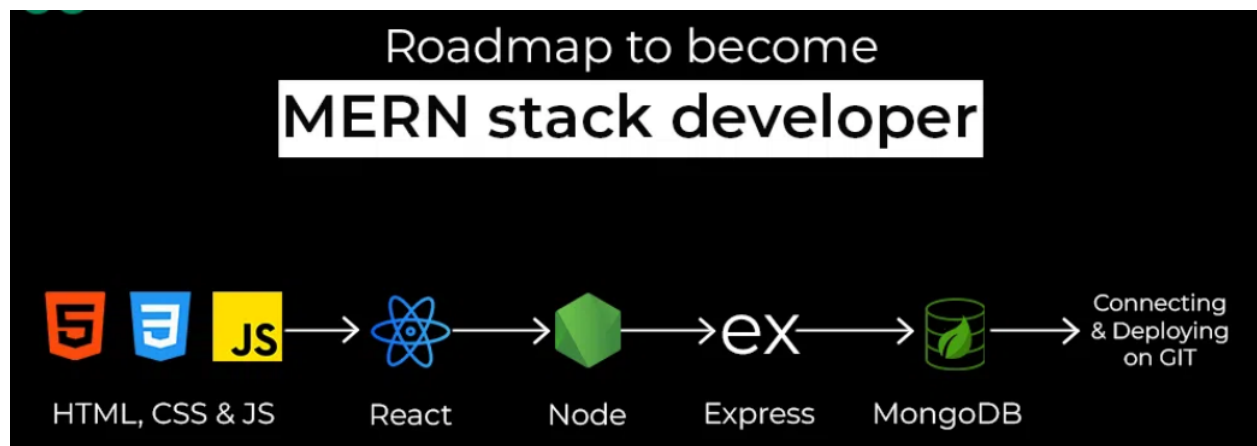
AGENDA

1	Intro to MERN Architecture
2	Setup MERN environment
	Installing and Testing Nodejs Npm package manager Installing Express Axios Mongoose by npm Installing and Testing MongoDB
3	Creating and running first Nodejs Hello App
4	Creating a Demo Site
5	IMAGE Display from a Folder
6	Mongo Database connectivity – Read write to database
7	Student Registration

What Is the MERN Stack?

The MERN stack is a collection of technologies that help developers build robust and scalable web applications using JavaScript. The acronym “MERN” stands for MongoDB, Express, React, and Node.js, with each component playing a role in the development process. MongoDB serves as a document-oriented database that can efficiently store data in JSON format. Express is a web application framework that provides a set of features to streamline the assembly of web applications. React is a front-end JavaScript library that offers a large toolkit for crafting user interfaces. Node.js is the runtime environment for execution of JavaScript code on the server side, coordinating communication between front-end and back-end components. As a group, these components make the MERN stack a preferred choice for developers seeking an efficient, full-stack development framework.

The MERN stack is ideal for developers who want to use a single language, JavaScript, for both the front end and back end of their applications. The MERN stack uses a collection of popular open source, cloud native technologies to support all steps in the web application’s functionality.



MERN is a pre-built technology stack based on JavaScript technologies. MERN stands for **M**ongoDB, **E**xpress, **R**ead, and **N**ode, after the four key technologies that make up the stack.

- MongoDB — document database
- Express(.js) — Node.js web framework
- React(.js) — a client-side JavaScript framework
- Node(.js) — the premier JavaScript web server (runtime)

Express and Node make up the middle (application) tier. Express.js is a server-side web framework, and Node.js is the popular and powerful JavaScript server platform. Regardless of which variant you choose, ME(RVA)N is the ideal approach to working with JavaScript and JSON, all the way through.

MERN Stack vs. MEAN Stack

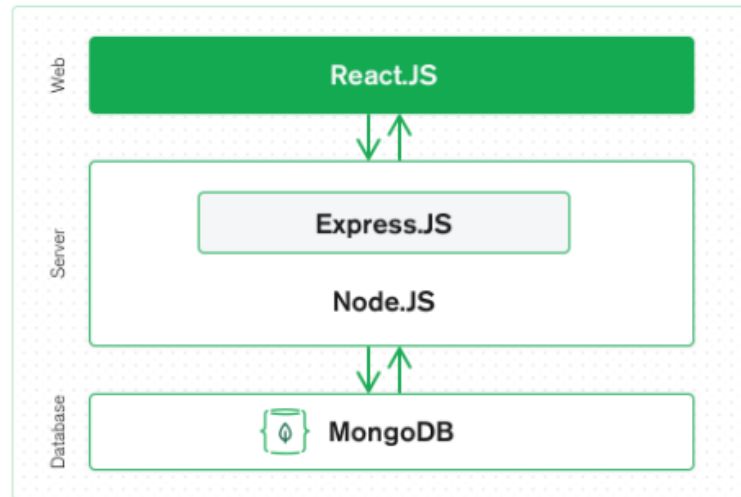
The MERN stack and the MEAN stack are both collections of popular JavaScript-based technologies used for web development. MEAN is an equivalent set of tools to MERN, simply replacing the React library and framework with another popular framework called Angular, also known as Angular.js. Both Angular and React are front-end technologies; the rest of the components stay the same.

MERN Stack Components

The MERN stack is a collection of JavaScript-based technologies that are used together to develop web applications. The stack consists of MongoDB, Express, React, and Node.js.

- MongoDB is a highly scalable document database that makes it easy to store and retrieve data in JSON documents.
- Express is a lightweight web application framework that provides a range of app-building tools and supports a variety of programming languages, including JavaScript.
- React is an open source, front-end JavaScript library for building user interfaces based on components.

- Node.js is a runtime environment that can be used to run JavaScript code on the server side. This allows developers to use the same language for both the front and back ends of their applications.

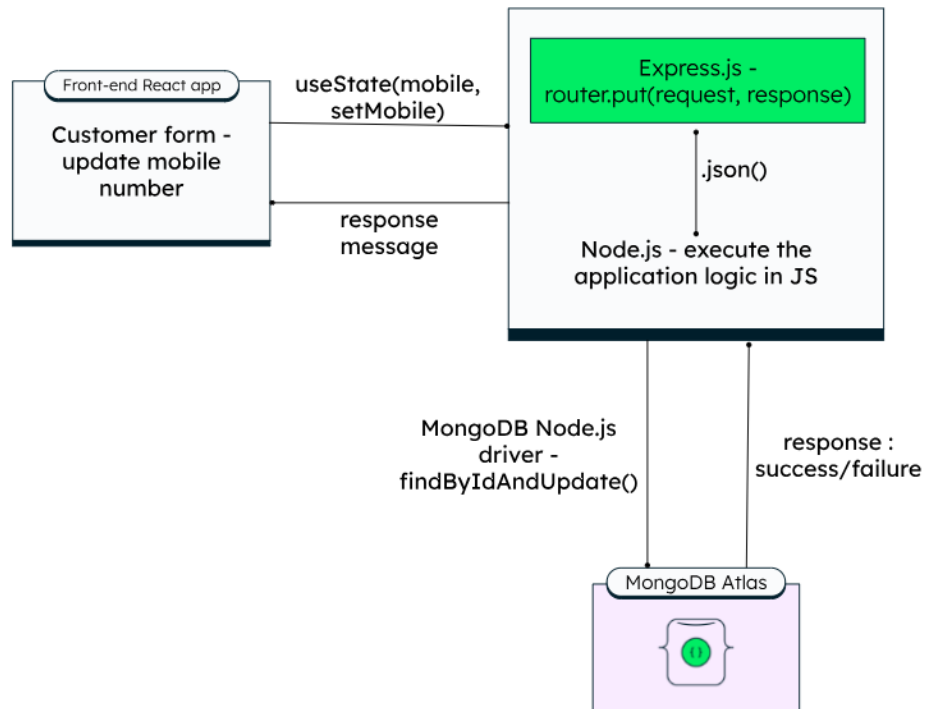


MERN Stack Feature and Use Cases

The MERN stack is giving rise to applications that are user-friendly and efficient, including the following:

- **Real-time applications:** The “N” in MERN stands for Node.js, which is ideal for handling real-time data streams in sites that feature collaboration tools, chat applications, and even gaming apps.
- **Single-page applications (SPAs):** SPAs are web applications that load a single HTML page and dynamically update it without reloading the entire page. This feature is enabled by the “R” in MERN, React, a popular library of JavaScript components that makes it easy to build updatable SPAs, such as maps, web email services, and social media pages.
- **Data-driven applications:** The “M” in MERN stands for MongoDB, a NoSQL database for storing and retrieving large amounts of data in document format. Express, the “E” in MERN, is an extremely efficient framework for managing interactions between the front end and the database. Examples of data-driven applications include social media applications, ecommerce applications, and content management systems.

Although developers often choose the MERN stack to build small, efficient web apps for tasks such as expense tracking and calendars, it also has been used to build mobile versions of big, high-traffic sites, such as Facebook, Instagram, WhatsApp, Dropbox, and Netflix.



When to choose the MERN stack?

If your project timelines are strict and requirements are well-defined, MERN stack is an ideal choice, that would save time and cost, and help developers get onboard quickly, as they have to focus on learning only one technology. Also, long-term maintenance of a project could be easier with MERN stack due to structured, well-established approach and extensive documentation.

MERN stack works well for any possible use case or project requirement that you have, as all of the components (MERN) offer powerful capabilities and features. The only time when you cannot choose MERN stack is when you want to use technologies other than JavaScript.

MERN – Environment setup

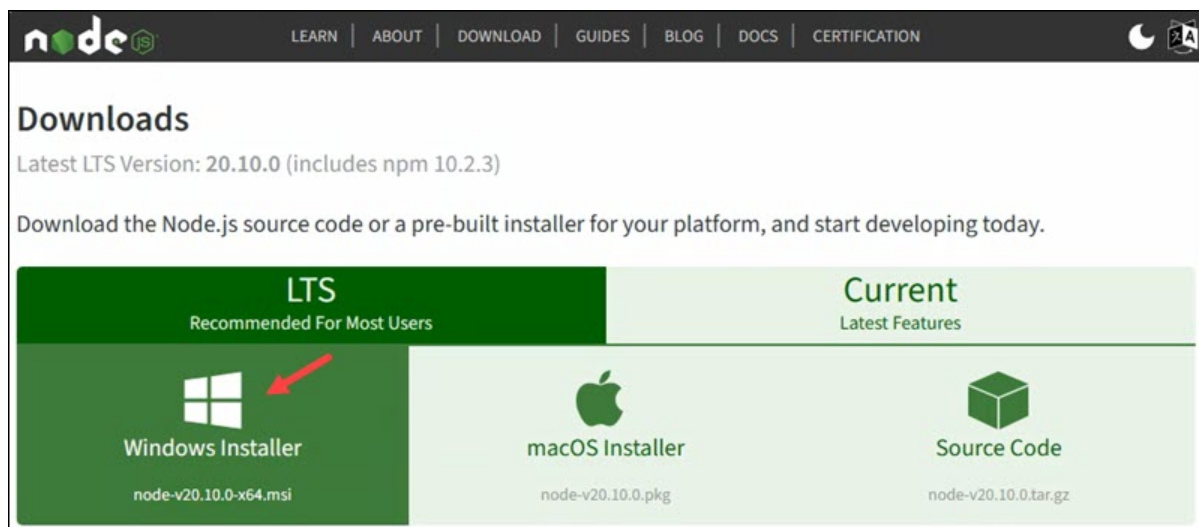
Installing Nodejs

Node.js is a runtime environment that includes everything you need to execute a program written in JavaScript. It's used for running scripts on servers to render content before it is delivered to a web browser.

Node **P**ackage **M**anager (**NPM**) is an application manager and repository for developing and sharing JavaScript code. It enables managing Node.js dependencies.

Step 1: Download Node.js Installer

In a web browser, navigate to the [Node.js Downloads](#) page. Click the **Windows Installer** button to download the latest stable version with long-term support (LTS). The installer also includes the NPM [package manager](#).



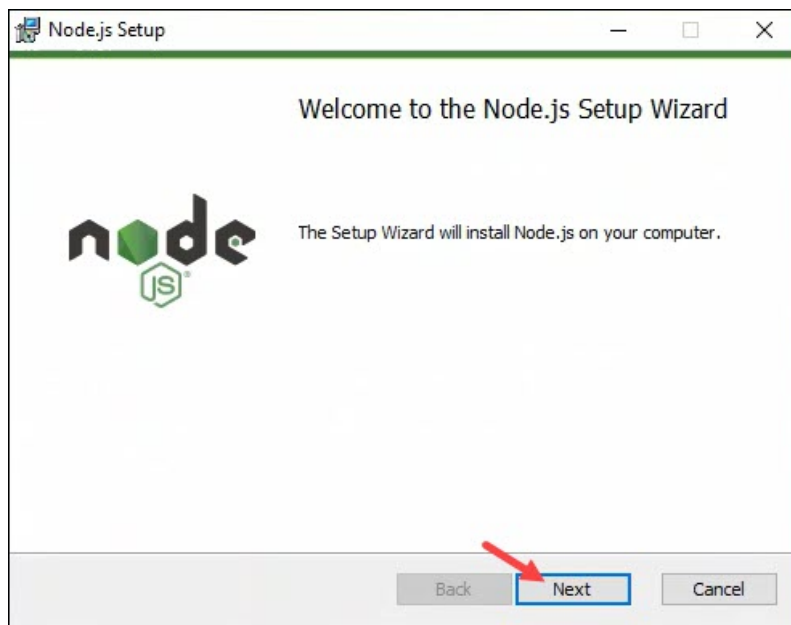
The [file](#) is saved in the **Downloads** folder by default.

Other versions of Node.js and NPM are available, so choose the appropriate one for your system and use case. Use the top tabs to switch from the LTS to the current version to test the newest features. If you are new to Node.js or don't need a specific version, choose LTS since it is tested and stable.

Step 2: Install Node.js and NPM

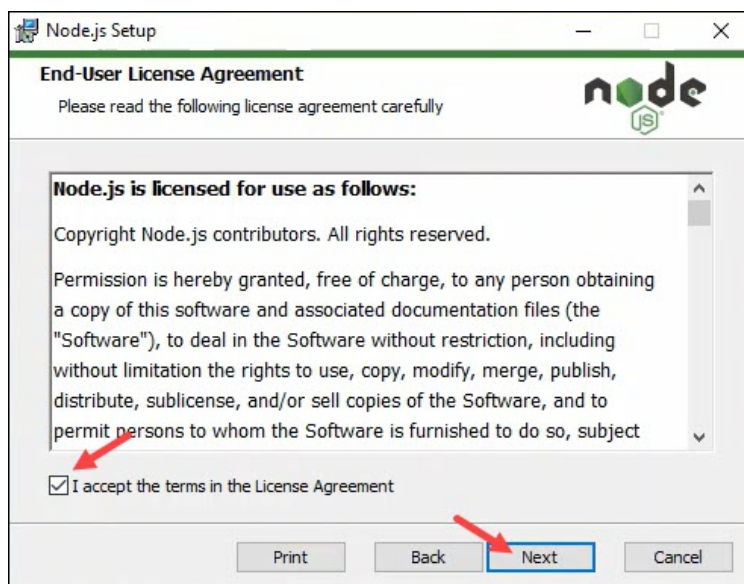
After downloading the installer, follow the steps below:

1. Launch the installer by double-clicking the downloaded file.
2. The Node.js Setup Wizard starts with the welcome screen.



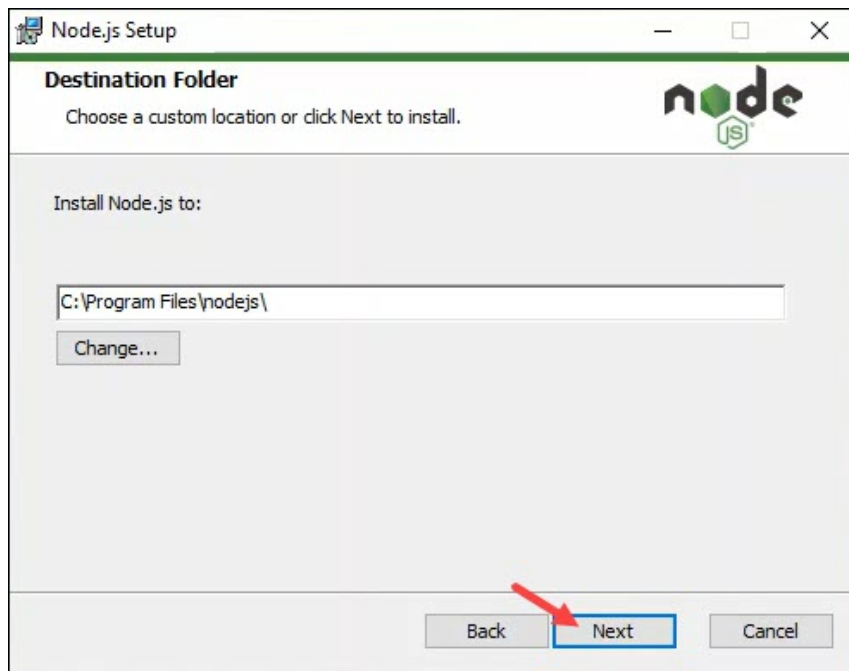
Click **Next** to proceed.

4. Review the end-user license agreement and click the checkbox to accept the terms and conditions.



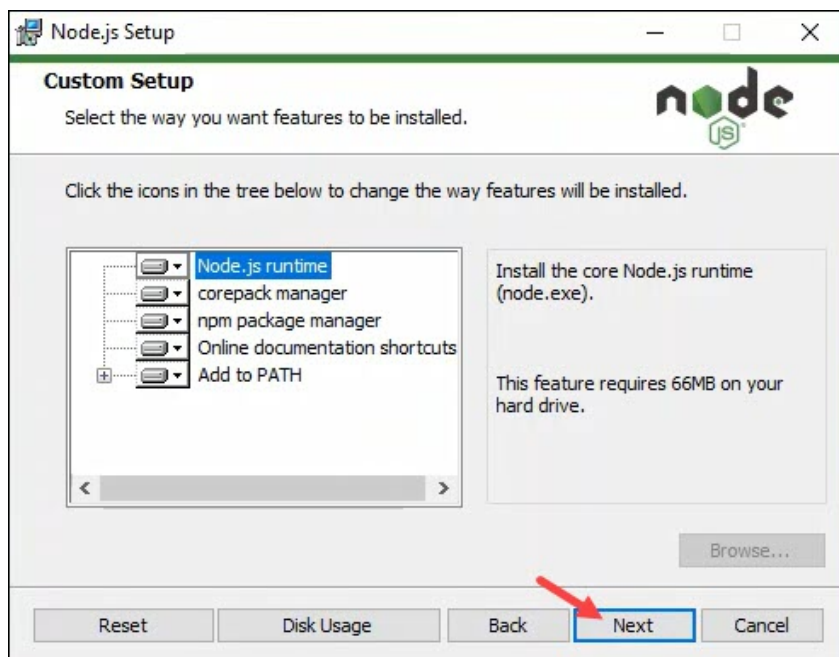
Click **Next** to continue.

5. The installer asks to choose the installation location.



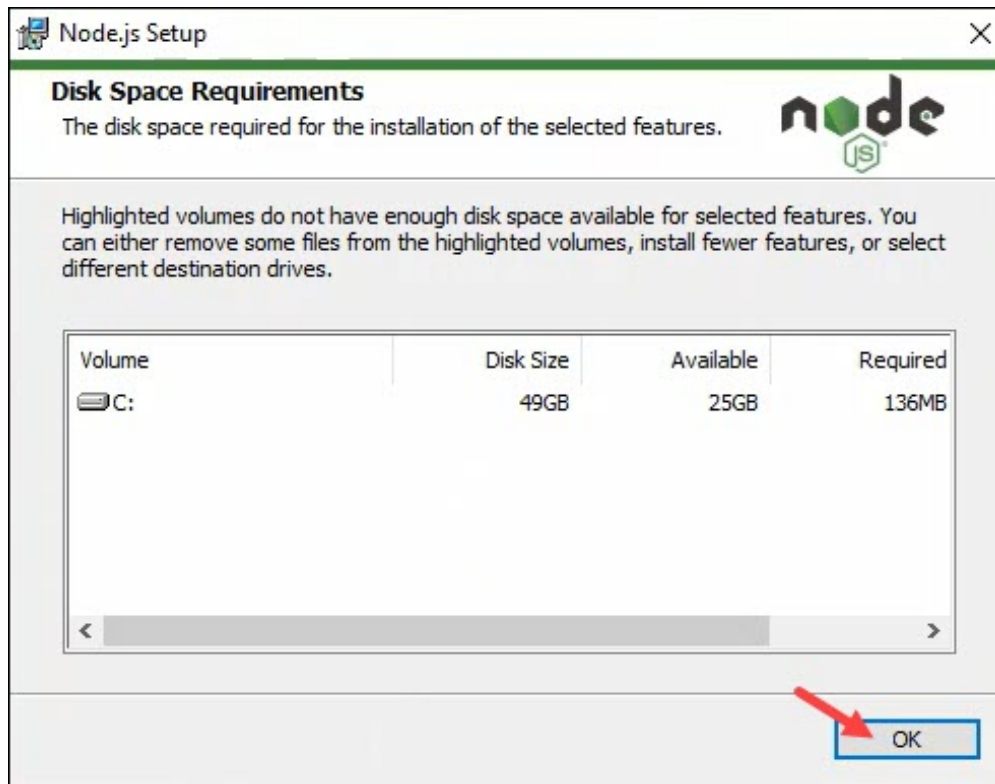
Leave the default location for a standard installation and click **Next** to proceed.

6. Select components to include or remove from the installation. The default options install Node.js, NPM, corepack, online documentation shortcuts, and add the programs to PATH.



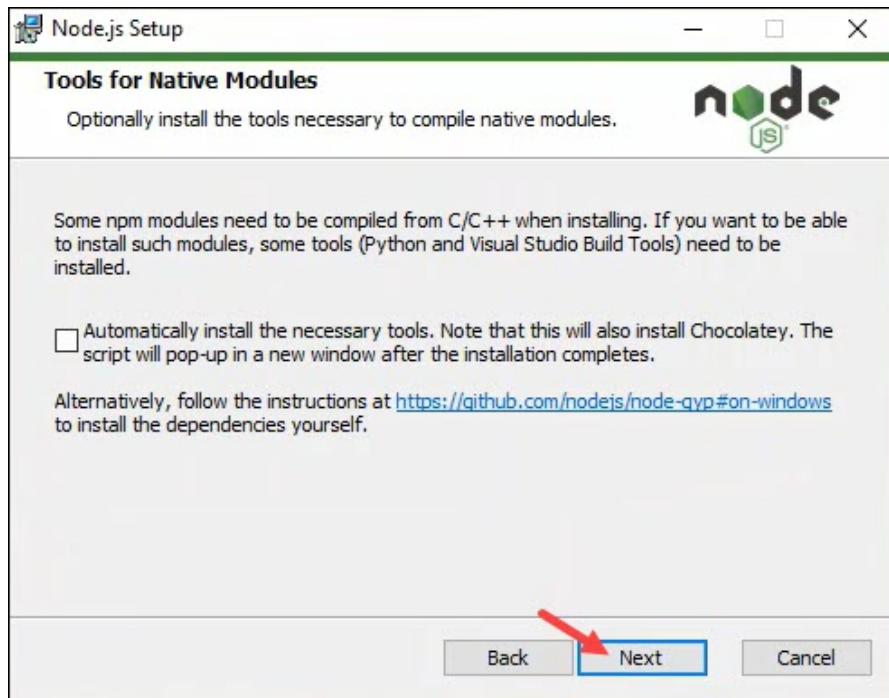
Customize the setup or click **Next** to accept the default values.

7. The following section shows the total required space for the installation and the available space on disk.



Click **OK** to proceed. Select a different disk or install fewer features if the installation does not allow proceeding.

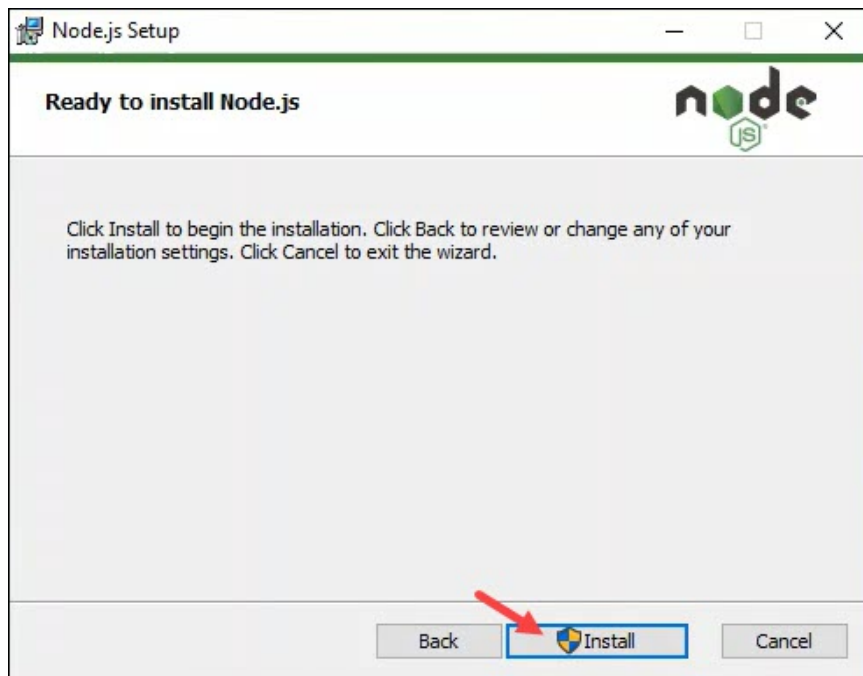
8. Choose whether to install additional dependencies for compiling native modules. Some NPM modules compile from C/C++ and require additional tools to function correctly (Python, Visual Studio Build Tools, and Chocolatey).



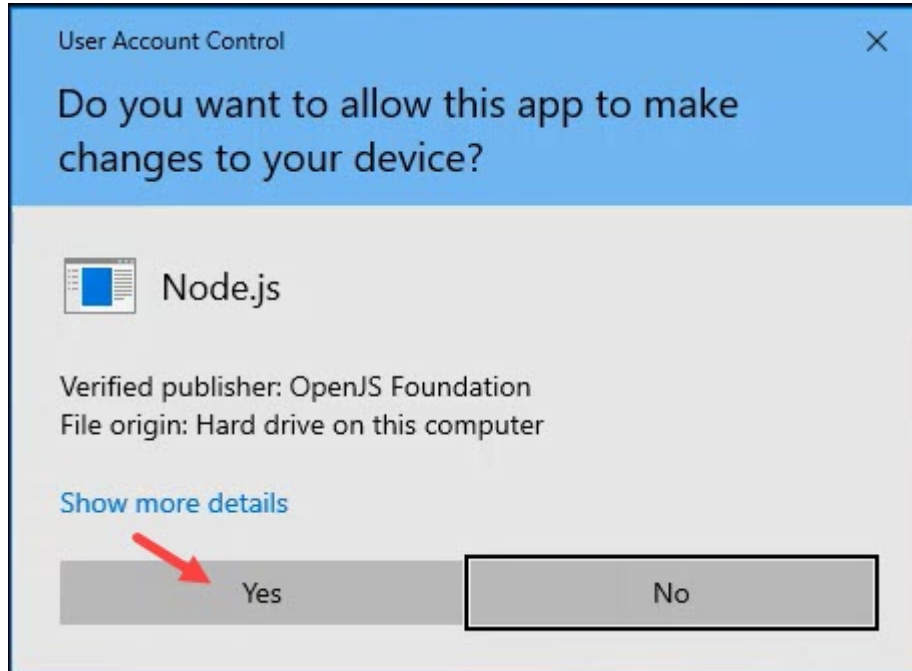
If you use such modules, select the checkbox and click **Next**. The selection of this option starts an installation script after the Node.js installation is complete.

For a simple installation, skip this step and click **Next** to proceed.

7. Click the **Install** button to start the installation.

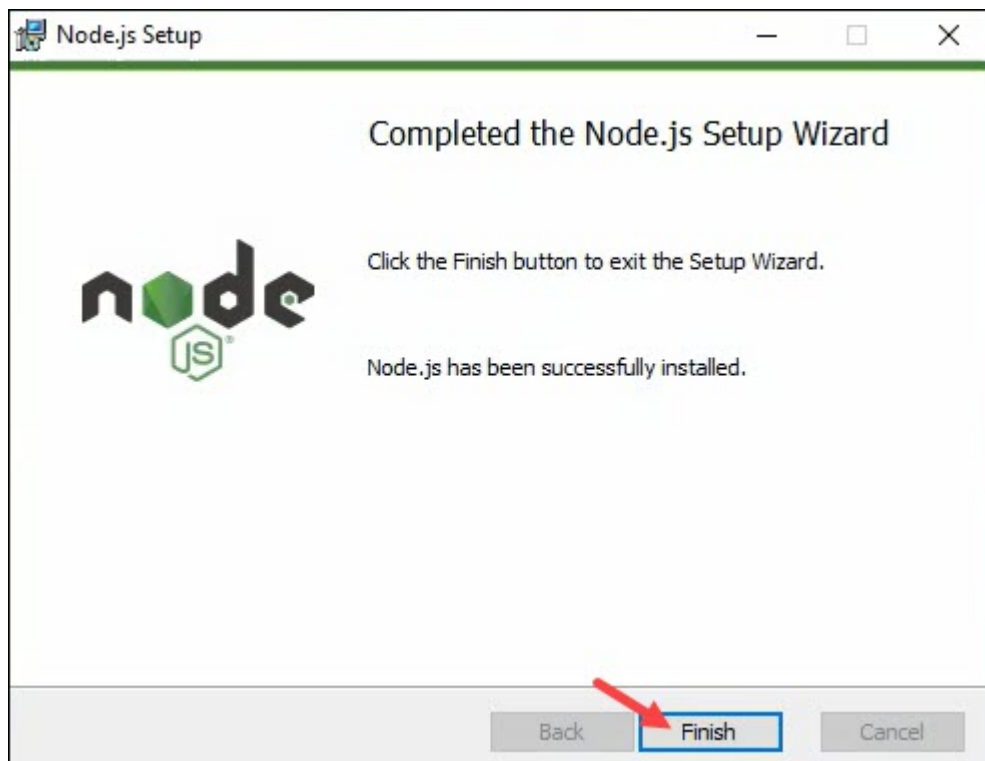


8. The installer prompts for administrator confirmation to make changes to the device.



Enter the administrator password if prompted and click **Yes** to continue.

9. The installation takes some time. When it is complete, the final screen shows a success message.



Click **Finish** to complete the installation and close the installer.

Note: If you selected to install native tools in step 8, closing the installer automatically starts a PowerShell script to install the listed tools.

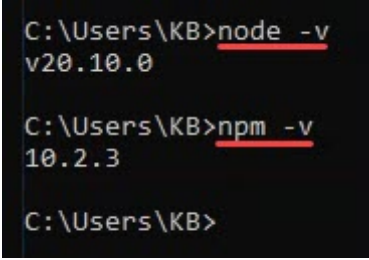
Step 3: Verify Installation

To verify Node.js installed successfully, run the following command in a command prompt :

```
node -v
```

The command shows the Node.js version installed on your system. Use the following command to check for NPM:

```
npm -v
```



```
C:\Users\KB>node -v
v20.10.0

C:\Users\KB>npm -v
10.2.3

C:\Users\KB>
```

Install MongoDB Community Edition

1

Download the MongoDB Community `.msi` installer from the internet link:

➤ MongoDB Download Center

- a. In the **Version** dropdown, select the version of MongoDB to download.
- b. In the **Platform** dropdown, select **Windows**.
- c. In the **Package** dropdown, select **msi**.
- d. Click **Download**.

2

Run the MongoDB installer.

For example, from the Windows Explorer/File Explorer:

- a. Go to the directory where you downloaded the MongoDB installer (`.msi` file). By default, this is your `Downloads` directory.
- b. Double-click the `.msi` file.

3

Follow the MongoDB Community Edition installation wizard.

The wizard steps you through the installation of MongoDB and MongoDB Compass.

Choose Setup Type

You can choose either the Complete (recommended for most users) or Custom setup type. The Complete setup option installs MongoDB and the MongoDB tools to the default location. The Custom setup option allows you to specify which executables are installed and where.

Service Configuration

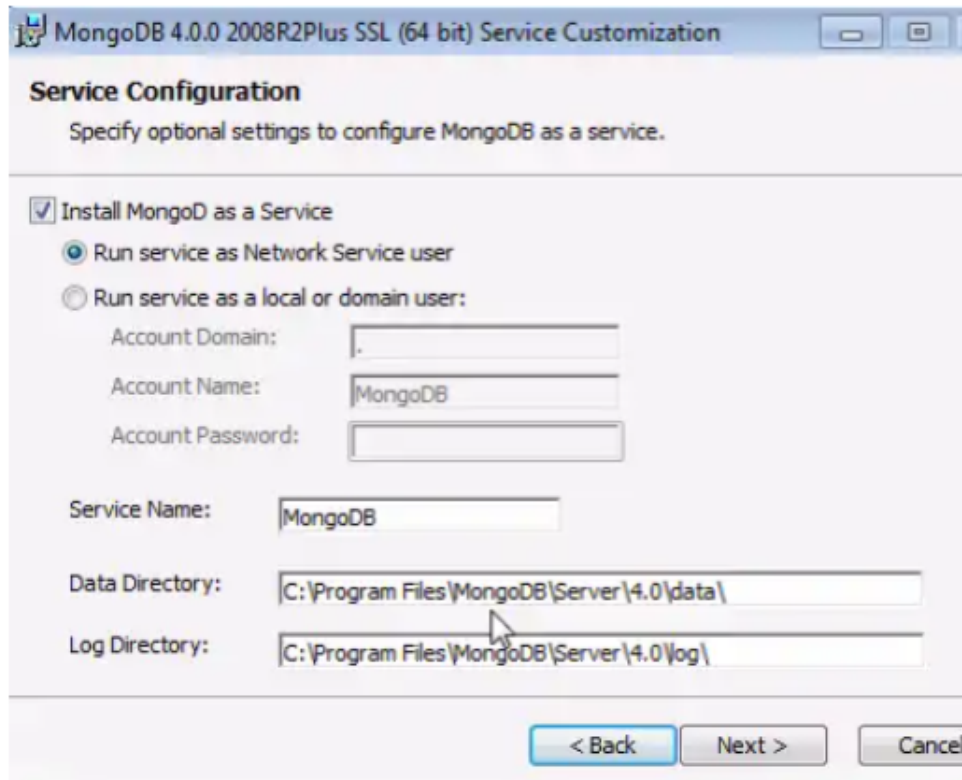
You can set up MongoDB as a Windows service during the install or just install the binaries.

MongoDB Service

MongoDB

The following installs and configures MongoDB as a Windows service.

You can configure and start MongoDB as a Windows service during the install, and the MongoDB service is started upon successful installation.



- Select **Install MongoDB as a Service** MongoDB as a service.
 - Select
 - **Run the service as Network Service user** (Default)

This is a Windows user account that is built-in to Windows

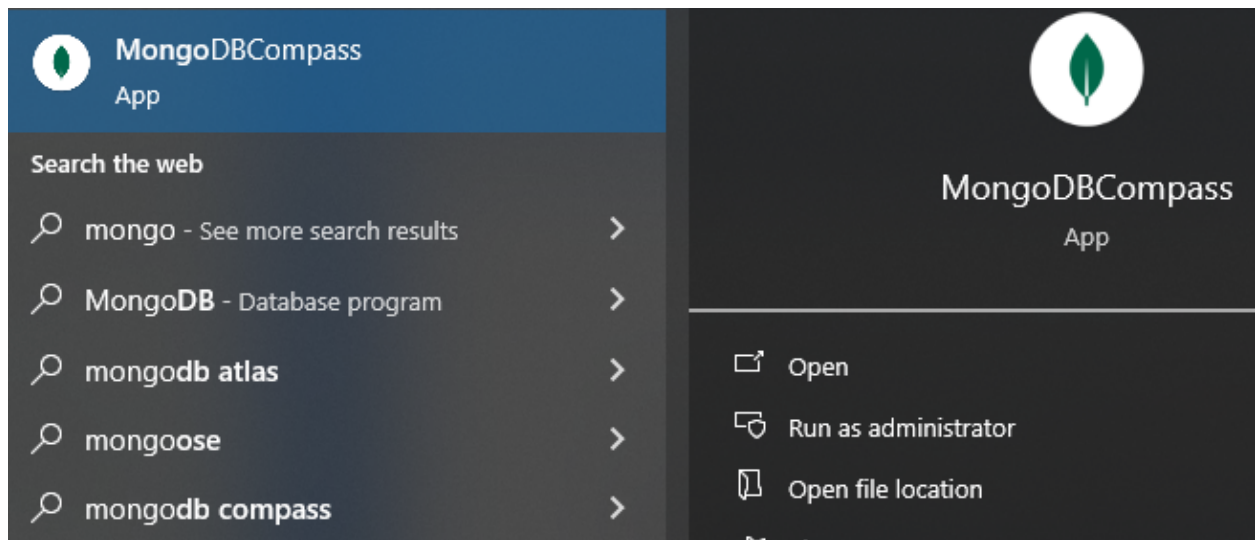
b. Install MongoDB Compass

Optional. To have the wizard install MongoDB Compass, select **Install MongoDB Compass** (Default).

c. When ready, click Install.

Testing MongoDB Installation

1- After installation open MongoDB from the windows search



The following Screen can be seen after you connect to the data base

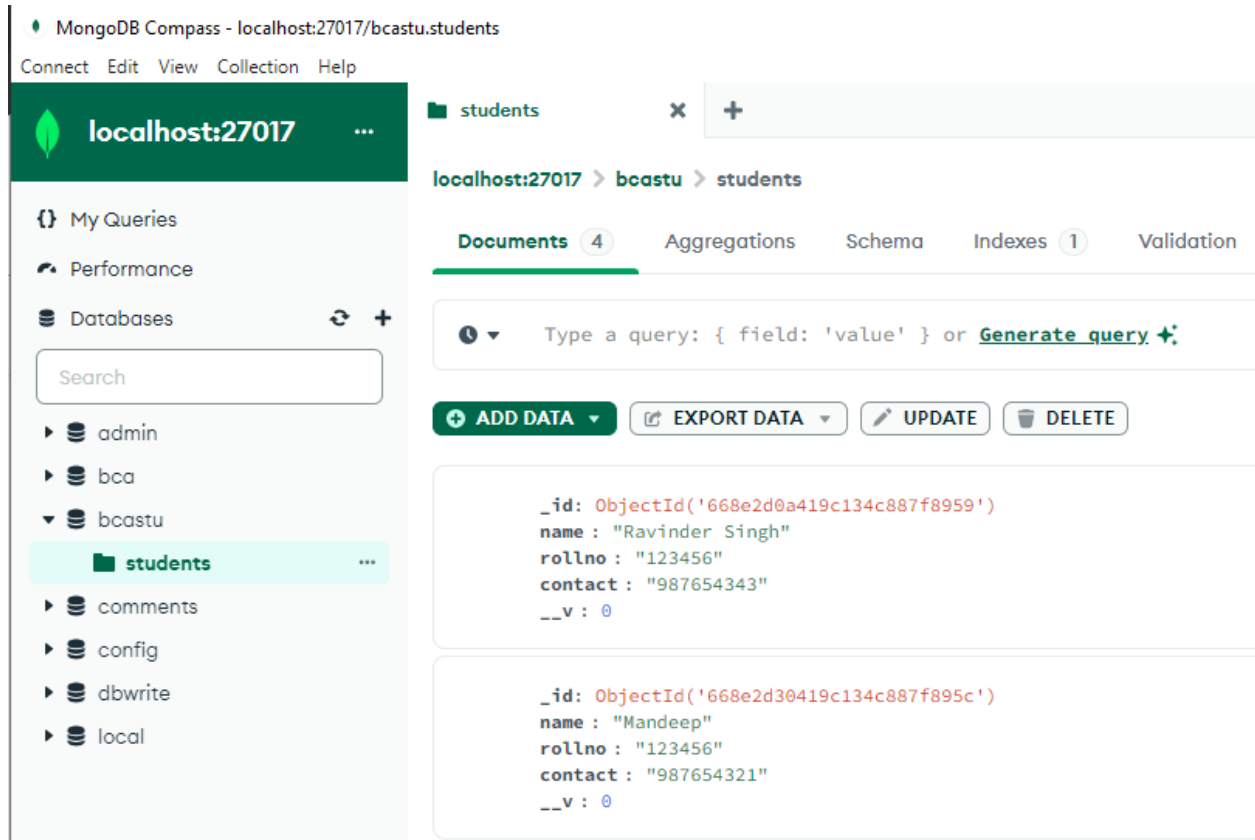


IMAGE Display from a Folder

- 1) Go to command window
- 2) `npx create-react-app imagedisplay`
- 3) open the folder imagedisplay in vscode
- 4) create a folder components in src folder
- 5) `src/components`
- 6) copy an image example.jpg in /public folder
- 7) write a script ImageDisplay.js in components folder to show the root page
- 8) change App.js file
- 9) in the command prompt give command
- 10) `npm start`

ImageDisplay.js

```
import React from 'react';

const ImageDisplay = () => {
  return (
    <div>
      <h1>Image Display</h1>
      <img src={`${process.env.PUBLIC_URL}/example.jpg`} alt="Example" style={{ maxWidth:
'100%', height: 'auto' }} />
    </div>
  );
}

export default ImageDisplay;
```

MERN – MongoDB – Database Connectivity

- 1) Install mongoDB with Compass
- 2) Start Compass and connect to Database
- 3) Create a directory dbwrite
- 4) Change to dbwrite directory `cd dbwrite`
- 5) Give command `npm init -y` (this will initialize the project folder)
- 6) Install package axios to interact with database
- 7) `npm install axios`
- 8) create a file in dbwrite folder that will read and write the text in database
- 9) Database with the name of the project is automatically created in mongodb
- 10) Create a file server.js in dbwrite folder that acts as a server and connects to the database
- 11) Start the server by the command window with `node server.js`
- 12) Open another command window and give command `readwrite.js` to read and write a text in the database
- 13) Check the data in mongodb with compass

Server.js

```
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');

const app = express();
const port = 3000;

// Middleware
app.use(bodyParser.json());

// MongoDB connection
mongoose.connect('mongodb://localhost:27017/dbwrite', {
```

```
    useNewUrlParser: true,
    useUnifiedTopology: true,
  }).then(() => {
    console.log('Connected to MongoDB');
  }).catch(err => {
    console.error('MongoDB connection error:', err);
  });

// Define a schema and model
const textSchema = new mongoose.Schema({
  content: String
});

const Text = mongoose.model('Text', textSchema);

// Routes
app.post('/texts', async (req, res) => {
  const newText = new Text({
    content: req.body.content
  });

  try {
    const savedText = await newText.save();
    res.status(201).json(savedText);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
});

app.get('/texts', async (req, res) => {
  try {
    const texts = await Text.find();
    res.json(texts);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
});

// Start the server
app.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}`);
});
```

Output

```
Command Prompt

]

C:\Users\Admin\bcatrg\dbwrite>node readwrite.js
Text saved: {
  content: 'Hello, GNIMT ...!',
  _id: '66852ad7fa3549edf2c2ef74',
  __v: 0
}
Texts retrieved: [
  { _id: '6684eebb2a1d1df092b86ba2', content: 'Hello, world!', __v: 0 },
  { _id: '6684fea12a1d1df092b86ba5', content: 'Hello, world!', __v: 0 },
  { _id: '66851668445416f19d71c5c1', content: 'Hello, world!', __v: 0 },
  { _id: '668516a0445416f19d71c5c4', content: 'Hello, world!', __v: 0 },
  { _id: '668517a9445416f19d71c5c8', content: 'Hello, world!', __v: 0 },
  { _id: '66851bd8fa3549edf2c2ef71', content: 'Hello, world!', __v: 0 },
  {
    _id: '66852ad7fa3549edf2c2ef74',
    content: 'Hello, GNIMT ...!',
    __v: 0
  }
]

C:\Users\Admin\bcatrg\dbwrite>node readwrite.js_
```

Readwrite.js

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

// Function to write text to the database
const writeTextToDatabase = async (text) => {
  try {
    const response = await axios.post('http://localhost:3000/texts', {
      content: text
    });
    console.log('Text saved:', response.data);
  } catch (error) {
    console.error('Error writing text to database:', error.message);
  }
};

// Function to read text from the database
const readTextFromDatabase = async () => {
  try {
    const response = await axios.get('http://localhost:3000/texts');
```

```
    console.log('Texts retrieved:', response.data);
  } catch (error) {
    console.error('Error reading text from database:', error.message);
  }
};

// Test the functions
const testDatabaseOperations = async () => {
  await writeTextToDatabase('Hello, GNIMT ...!');
  await readTextFromDatabase();
};

testDatabaseOperations();
```

Creating a Demo Site

Creating a simple React project named `demosite` with a home page, a header with a menu and logo, and a footer with social media links can be done in a few steps.

Below are the instructions and the necessary code for each component.

Steps to Create the React Project

1. Set Up the Project:

- Make sure you have Node.js and npm installed.
- Create a new React project using Create React App.

```
npx create-react-app demosite  
cd demosite
```

2. Create Components and Pages:

- Create components for `Header`, `Footer`, and pages for `Home`, `About`, `Faculty`, `Courses`, and `Contact`.

3. Add Routing:

- Use React Router to handle navigation between pages.

4. Add CSS:

- Create a CSS file to style the components.

Step-by-Step Implementation

1. Set Up the Project

Run the following commands to set up the project:

```
npx create-react-app demosite  
cd demosite  
npm install react-router-dom
```

2. Create the Components

src/components/Header.js

```
import React from 'react';  
import { Link } from 'react-router-dom';  
import './Header.css';
```

```

const Header = () => {
  return (
    <header>
      <div className="logo">MyLogo</div>
      <nav>
        <ul>
          <li><Link to="/">Home</Link></li>
          <li><Link to="/about">About</Link></li>
          <li><Link to="/faculty">Faculty</Link></li>
          <li><Link to="/courses">Courses</Link></li>
          <li><Link to="/contact">Contact</Link></li>
        </ul>
      </nav>
    </header>
  );
}
export default Header;

```

src/components/Footer.js

```

import React from 'react';
import './Footer.css';

const Footer = () => {
  return (
    <footer>
      <div>
        <a href="https://facebook.com">Facebook</a>
        <a href="https://twitter.com">Twitter</a>
        <a href="https://instagram.com">Instagram</a>
      </div>
    </footer>
  );
}
export default Footer;

```

src/components/Header.css

```

header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  background-color: #333;
  padding: 10px 20px;
}

```



```
.logo {  
  color: white;  
  font-size: 1.5em;  
}
```

```
nav ul {  
  display: flex;  
  list-style: none;  
}
```

```
nav ul li {  
  margin: 0 10px;  
}
```

```
nav ul li a {  
  color: white;  
  text-decoration: none;  
}
```

```
nav ul li a:hover {  
  text-decoration: underline;  
}
```

src/components/Footer.css

```
footer {  
  background-color: #333;  
  color: white;  
  text-align: center;  
  padding: 10px 0;  
  position: absolute;  
  bottom: 0;  
  width: 100%;  
}
```

```
footer a {  
  color: white;  
  margin: 0 10px;  
  text-decoration: none;  
}
```

```
footer a:hover {  
  text-decoration: underline;  
}
```

3. Create the Pages

src/pages/Home.js

```
import React from 'react';
const Home = () => {
  return <h1>Welcome to the Home Page</h1>;
}
export default Home;
```

src/pages/About.js

```
import React from 'react';
const About = () => {
  return <h1>About Us</h1>;
}

export default About;
```

src/pages/Faculty.js

```
import React from 'react';

const Faculty = () => {
  return <h1>Our Faculty</h1>;
}
export default Faculty;
```

src/pages/Courses.js

```
import React from 'react';

const Courses = () => {
  return <h1>Our Courses</h1>;
}
export default Courses;
```

src/pages/Contact.js

```
import React from 'react';

const Contact = () => {
  return <h1>Contact Us</h1>;
}
export default Contact;
```

4. Set Up Routing in App.js

src/App.js

```
import React from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import Header from './components/Header';
import Footer from './components/Footer';
import Home from './pages/Home';
import About from './pages/About';
import Faculty from './pages/Faculty';
import Courses from './pages/Courses';
import Contact from './pages/Contact';
import './App.css';

const App = () => {
  return (
    <Router>
      <div className="App">
        <Header />
        <div className="content">
          <Switch>
            <Route exact path="/" component={Home} />
            <Route path="/about" component={About} />
            <Route path="/faculty" component={Faculty} />
            <Route path="/courses" component={Courses} />
            <Route path="/contact" component={Contact} />
          </Switch>
        </div>
        <Footer />
      </div>
    </Router>
  );
}
```

```
export default App;
```

5. Add Global CSS

src/App.css

```
body, html {
  margin: 0;
  padding: 0;
```

```
    font-family: Arial, sans-serif;
}
```

```
.App {
  display: flex;
  flex-direction: column;
  min-height: 100vh;
}
```

```
.content {
  flex: 1;
}
```

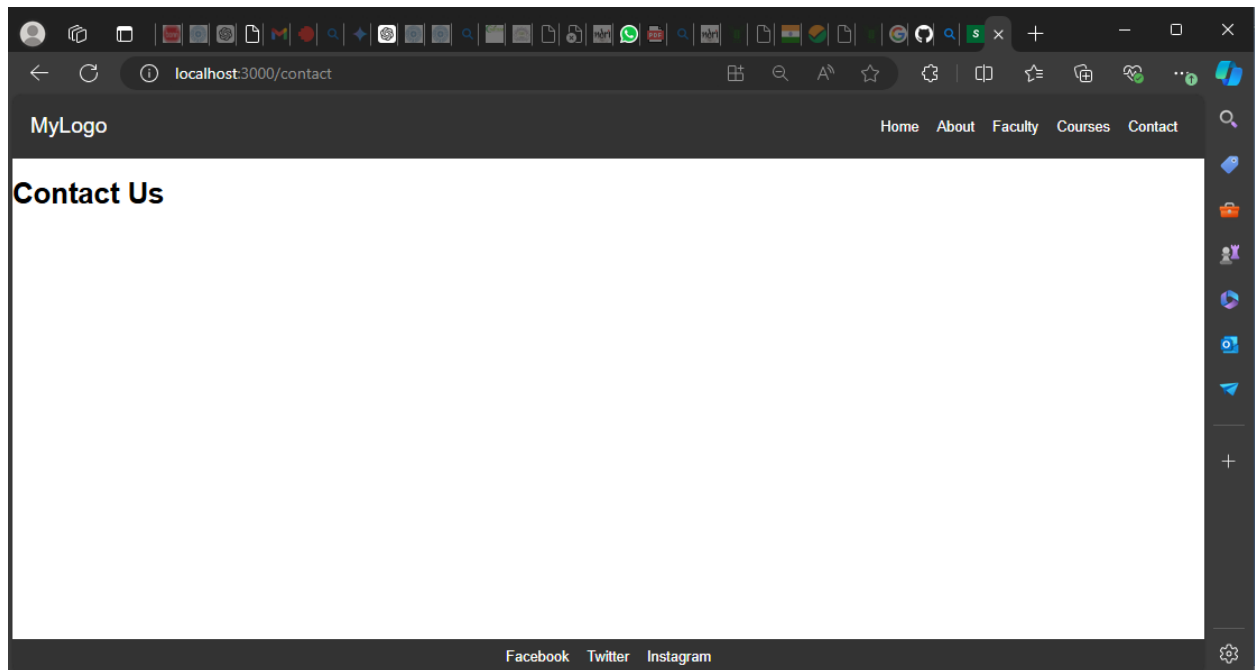
6. Run the Project

npm start

This will start the development server, and you can view your project at `http://localhost:3000`.

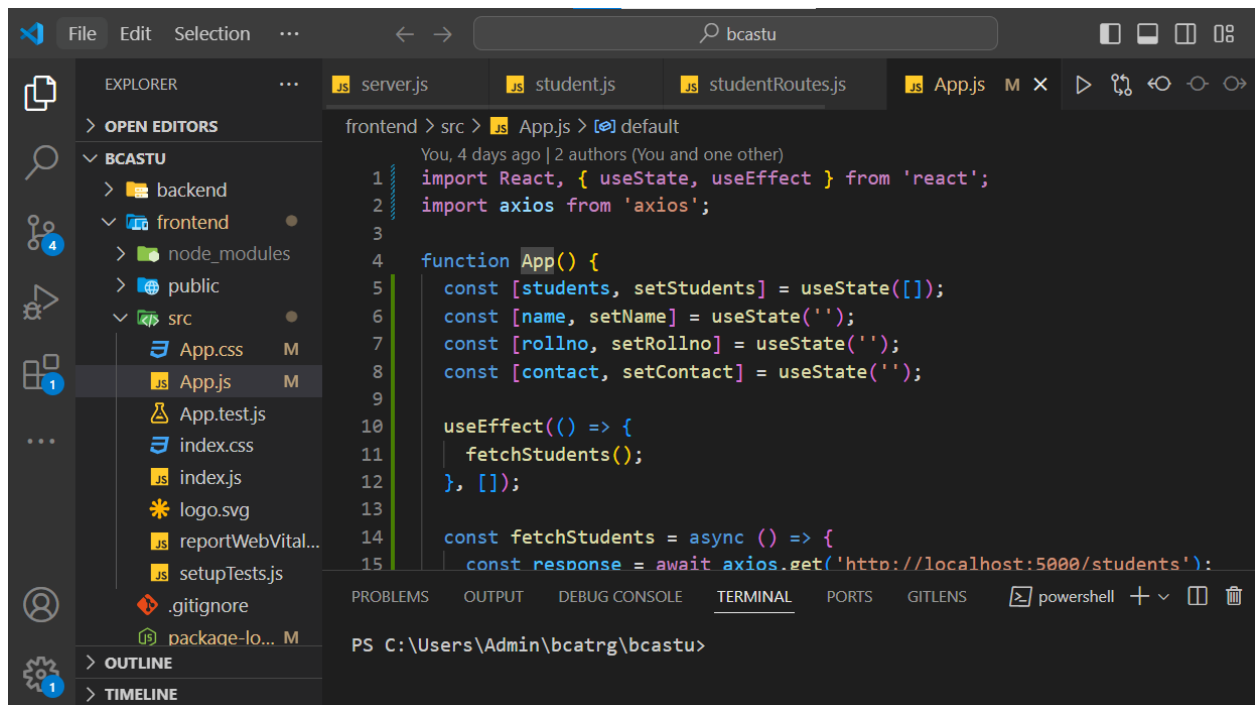
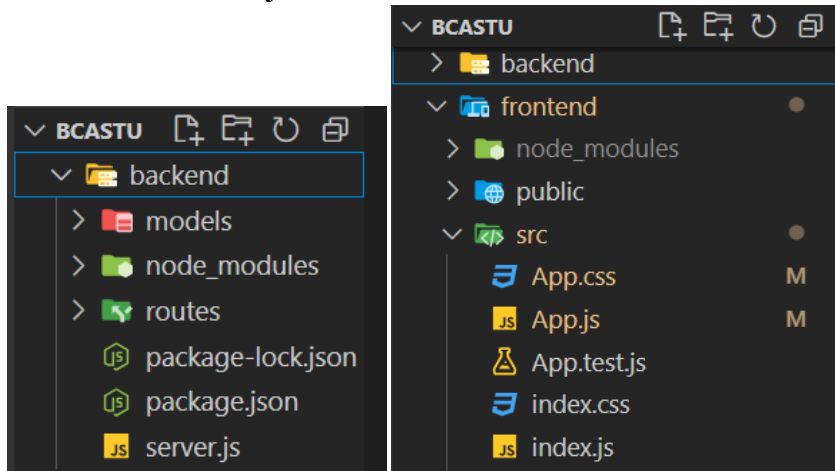
Your project structure should look like this:

```
demosite/
├── node_modules/
├── public/
├── src/
│   ├── components/
│   │   ├── Footer.js
│   │   ├── Footer.css
│   │   ├── Header.js
│   │   └── Header.css
│   ├── pages/
│   │   ├── About.js
│   │   ├── Contact.js
│   │   ├── Courses.js
│   │   ├── Faculty.js
│   │   └── Home.js
│   ├── App.css
│   ├── App.js
│   └── index.js
├── package.json
└── README.md
```



Student Registration

1. Create a folder bcastu and Go to bcastu folder `cd bcastu`
2. Create two more folders `frontend` and `backend`
3. Backend folder deals with database connects and interaction with server
4. Frontend folder deals with User interface , student data collection with form
5. Create folder `models` and `routes` in `backend`
6. Create folder `public` in `frontend` folder
7. Create a file `student.js` in `models`



Code

Student.js

```
> models > JS student.js > ...
const mongoose = require('mongoose');

const studentSchema = new mongoose.Schema({
  name: String,
  rollno: String,
  contact: String
});

module.exports = mongoose.model('Student', studentSchema);
```

StudentRoutes.js

```
d > routes > JS studentRoutes.js > ...
const express = require('express');
const router = express.Router();
const Student = require('../models/student');

// Create a student
router.post('/', async (req, res) => {
  const { name, rollno, contact } = req.body;
  const newStudent = new Student({ name, rollno, contact });
  await newStudent.save();
  res.json(newStudent);
});

// Get all students
router.get('/', async (req, res) => {
  const students = await Student.find();
  res.json(students);
});

module.exports = router;
```

Running the App

- 1- Go to backend folder `cd backend`
- 2- Issue the command `node server.js`, this will start the backend server

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

PS C:\Users\Admin\bcatr\bcastu> cd backend
PS C:\Users\Admin\bcatr\bcastu\backend> node server.js
(node:18812) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
(Use `node --trace-warnings ...` to show where the warning was created)
(node:18812) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
Server running on port 5000
MongoDB connected
```

- 3- Now open separate command window
- 4- go to frontend folder `cd frontend`
- 5- Issue command `npm start`, this will start the development server

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

PS C:\Users\Admin\bcatr\bcastu> cd frontend
Compiled successfully!

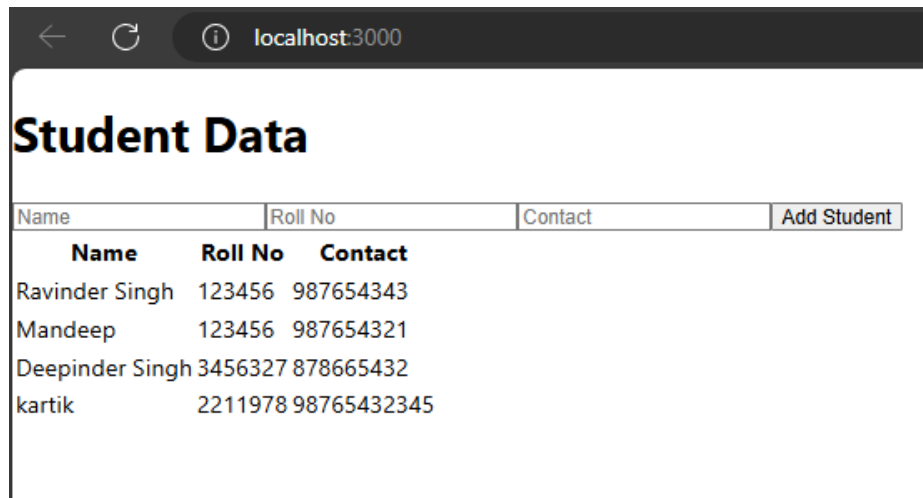
You can now view frontend in the browser.

Local:      http://localhost:3000
On Your Network:  http://192.168.56.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```


6- Web browser opens with student registration form



The screenshot shows a web browser window with the address bar displaying 'localhost:3000'. The page title is 'Student Data'. Below the title, there is a table with four columns: 'Name', 'Roll No', 'Contact', and 'Add Student'. The table contains four rows of student data.

Name	Roll No	Contact	Add Student
Ravinder Singh	123456	987654343	
Mandeep	123456	987654321	
Deepinder Singh	3456327	878665432	
kartik	2211978	98765432345	