For express: npm install express

For mongodb: npm install mongodb

For react: npx create-react-app my-app

New node.js project: npm init -y

React and react DOM: npm install react react-dom

Registration form using react:

```
import React, { Component } from 'react';

class RegistrationForm extends Component {
 constructor(props) {
  super(props);
  this.state = {
   name: '',
   email: '',
   password: ''
  };
 }

 handleChange = (event) => {
  const { name, value } = event.target;
  this.setState({ [name]: value });
 };

 handleSubmit = (event) => {
  event.preventDefault();
```

```
  console.log('Form Submitted:', this.state);
  alert(` Registered Successfully!\nName: ${this.state.name}\nEmail: ${this.state.email}` );
  this.setState({ name: '', email: '', password: '' }); // Reset form
 };

 render() {
  return (
   <div style={{ textAlign: 'center', marginTop: '50px' }}>
    <h2>Registration Form</h2>
    <form onSubmit={this.handleSubmit} style={{ display: 'inline-block', textAlign: 'left' }}>
     <div style={{ marginBottom: '10px' }}>
      <label>
       Name:
       <input
        type="text"
        name="name"
        value={this.state.name}
        onChange={this.handleChange}
        style={{ marginLeft: '10px' }}
        required
       />
      </label>
     </div>
     <div style={{ marginBottom: '10px' }}>
      <label>
```

```
    Email:
    <input
     type="email"
     name="email"
     value={this.state.email}
     onChange={this.handleChange}
     style={{ marginLeft: '10px' }}
     required
    />
   </label>
  </div>
  <div style={{ marginBottom: '10px' }}>
   <label>
    Password:
    <input
     type="password"
     name="password"
     value={this.state.password}
     onChange={this.handleChange}
     style={{ marginLeft: '10px' }}
     required
    />
   </label>
  </div>
  <button type="submit" style={{ padding: '5px 10px' }}>Register</button>
 </form>
```

```
    </div>
   );
  }
}

export default RegistrationForm;
```

2. registration form using function

```
import React, { useState } from 'react';

function RegistrationForm() {
  const [formData, setFormData] = useState({
    name: '',
    email: '',
    password: ''
  });

  const handleChange = (event) => {
    const { name, value } = event.target;
    setFormData({ ...formData, [name]: value });
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    console.log('Form Submitted:', formData);
```

```
  alert(` Registered Successfully!\nName: ${formData.name}\nEmail: ${formData.email}` );
  setFormData({ name: '', email: '', password: '' }); // Reset form
 };

 return (
  <div style={{ textAlign: 'center', marginTop: '50px' }}>
   <h2>Registration Form</h2>
   <form onSubmit={handleSubmit} style={{ display: 'inline-block', textAlign: 'left' }}>
    <div style={{ marginBottom: '10px' }}>
     <label>
      Name:
      <input
       type="text"
       name="name"
       value={formData.name}
       onChange={handleChange}
       style={{ marginLeft: '10px' }}
       required
      />
     </label>
    </div>
    <div style={{ marginBottom: '10px' }}>
     <label>
      Email:
      <input
```

```
          type="email"
          name="email"
          value={formData.email}
          onChange={handleChange}
          style={{ marginLeft: '10px' }}
          required
         />
        </label>
      </div>
      <div style={{ marginBottom: '10px' }}>
        <label>
         Password:
         <input
          type="password"
          name="password"
          value={formData.password}
          onChange={handleChange}
          style={{ marginLeft: '10px' }}
          required
         />
        </label>
      </div>
      <button type="submit" style={{ padding: '5px 10px' }}>Register</button>
    </form>
  </div>
);
```

```
}

export default RegistrationForm;
```

program for applying css style in react.js:

```
/* App.css */
.container {
  text-align: center;
  margin-top: 50px;
}

.heading {
  font-size: 24px;
  color: #4CAF50;
}

.form {
  display: inline-block;
  text-align: left;
}

.input {
  margin-left: 10px;
```

```
  padding: 5px;
  width: 200px;
}

.button {
  padding: 10px 20px;
  background-color: #4CAF50;
  color: white;
  border: none;
  cursor: pointer;
}

.button:hover {
  background-color: #45a049;
}
```

App.js update:

```
import React, { useState } from 'react';
import './App.css';

function App() {
  const [formData, setFormData] = useState({ name: '', email: '' });

  const handleChange = (event) => {
    const { name, value } = event.target;
    setFormData({ ...formData, [name]: value });
  };
```

```
const handleSubmit = (event) => {
  event.preventDefault();
  alert(` Submitted: Name - ${formData.name}, Email - ${formData.email}` );
  setFormData({ name: '', email: '' });
};

// Inline styles
const headingStyle = {
  fontSize: '20px',
  color: '#333',
  margin: '20px 0',
};

return (
  <div className="container">
    <h1 className="heading">React Form with CSS Styling</h1>
    <h2 style={headingStyle}>Inline Styled Subheading</h2>
    <form onSubmit={handleSubmit} className="form">
      <div style={{ marginBottom: '10px' }}>
        <label>
          Name:
          <input
            type="text"
            name="name"
            value={formData.name}
```

```
            onChange={handleChange}
            className="input"
            required
           />
          </label>
        </div>
        <div style={{ marginBottom: '10px' }}>
         <label>
          Email:
          <input
           type="email"
           name="email"
           value={formData.email}
           onChange={handleChange}
           className="input"
           required
          />
         </label>
        </div>
        <button type="submit" className="button">
         Submit
        </button>
      </form>
    </div>
  );
}
```

```
export default App;
```

5 MUI Components:

```
import React from 'react';
import { Button, Typography, TextField, Card, CardContent } from '@mui/material';
import '@fontsource/roboto'; // Optional: For Material UI Typography styling

function MUIComponentsDemo() {
  return (
    <div style={{ textAlign: 'center', marginTop: '50px' }}>
      {/* Typography Component */}
      <Typography variant="h4" component="h1" gutterBottom>
        Material-UI Components Demo
      </Typography>

      {/* Button Component */}
      <Button variant="contained" color="primary" style={{ margin: '10px' }}>
        Primary Button
      </Button>
      <Button variant="outlined" color="secondary" style={{ margin: '10px' }}>
        Secondary Button
      </Button>

      {/* TextField Component */}
      <div style={{ margin: '20px 0' }}>
```

```
      <TextField
        label="Enter Text"
        variant="outlined"
        color="primary"
        style={{ width: '300px' }}
      />
    </div>

    {/* Card Component */}
    <Card style={{ maxWidth: '400px', margin: '0 auto', padding: '20px' }}>
      <CardContent>
        <Typography variant="h6" gutterBottom>
          Card Component
        </Typography>
        <Typography variant="body2" color="textSecondary">
          This is an example of a Material-UI Card component with some text inside.
        </Typography>
      </CardContent>
    </Card>

    {/* Typography Component with another variant */}
    <Typography variant="caption" display="block" style={{ marginTop: '20px' }}>
      Powered by Material-UI
    </Typography>
  </div>
);
```

```
}

export default MUIComponentsDemo;
```

Node.JS web module hello message:

```
// Import the HTTP module
const http = require('http');

// Create a server
const server = http.createServer((req, res) => {
  // Set the response header with a status code and content type
  res.writeHead(200, { 'Content-Type': 'text/plain' });

  // Send the response body
  res.end('Hello, World!');
});

// Define the port number
const PORT = 3000;

// Start the server
server.listen(PORT, () => {
  console.log(` Server running at http://localhost:${PORT}/ `);
});
```

If you encounter a command not found error for node or npm, it might be due to your system's PATH configuration. Ensure the Node.js installation directory is added to your system’s PATH environment variable.For most installations, this is automatically handled by the Node.js installer.

// A simple program to demonstrate the concept of callback functions in Node.js

```
// A function that simulates a time-consuming task using setTimeout
function fetchData(callback) {
  console.log("Fetching data, please wait...");

  // Simulate a delay (e.g., fetching data from a database or API)
  setTimeout(() => {
    const data = { id: 1, name: "Node.js Callback Example" };
    console.log("Data fetched successfully!");

    // Call the callback function with the fetched data
    callback(data);
  }, 2000); // 2-second delay
}

// A function to process the data
function processData(data) {
  console.log("Processing Data:", data);
  console.log(` Processed Name: ${data.name.toUpperCase()}` );
}
```

```
// Main program execution
console.log("Starting the program...");
fetchData(processData);
console.log("Program continues while data is being fetched...");
```

program to read the file contents using Node.js file system.

// A simple program to demonstrate the concept of reading file contents using Node.js file system module

```
// Import the fs (File System) module
const fs = require('fs');

// Path to the file to be read
const filePath = './example.txt';

// Read the file asynchronously
fs.readFile(filePath, 'utf8', (err, data) => {
  if (err) {
    console.error("Error reading the file:", err.message);
    return;
  }
  console.log("File Contents:\n", data);
});

// Main program execution
console.log("Reading the file...");
```

Create a file named **example.txt** in the same directory as your script and add some text to it.

Save the script as readFileDemo.js.

Run the program:

node readFileDemo.js

program to write the contents to the file using Node.js file system.

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

// Data to be written to the file
const content = 'Hello, this is a test content written to the file!';

// Writing content to the file
fs.writeFile('output.txt', content, (err) => {
  if (err) {
    console.error('Error writing to the file:', err);
  } else {
    console.log('Content successfully written to the file!');
  }
});
```

program to read the contents from the directory and display on console using Node.js.

```
const fs = require('fs');
const path = './'; // specify the directory path, here it's the current directory
```

```
// Reading the contents of the directory
fs.readdir(path, (err, files) => {
  if (err) {
    console.error('Error reading the directory:', err);
  } else {
    console.log('Files in the directory:');
    files.forEach(file => {
      console.log(file);
    });
  }
});
```

5 functions of file system:

```
const fs = require('fs');
const path = './'; // Specify the directory path, here it's the current directory

// 1. Writing data to a file (fs.writeFile)
fs.writeFile('example.txt', 'Hello, Node.js!', (err) => {
  if (err) {
    console.error('Error writing to the file:', err);
  } else {
    console.log('Data successfully written to example.txt');
  }
});

// 2. Reading file contents (fs.readFile)
```

```
fs.readFile('example.txt', 'utf8', (err, data) => {
  if (err) {
    console.error('Error reading the file:', err);
  } else {
    console.log('File content:', data);
  }
});

// 3. Checking if a file or directory exists (fs.existsSync)
if (fs.existsSync('example.txt')) {
  console.log('example.txt exists!');
} else {
  console.log('example.txt does not exist!');
}

// 4. Renaming a file (fs.rename)
fs.rename('example.txt', 'renamed_example.txt', (err) => {
  if (err) {
    console.error('Error renaming the file:', err);
  } else {
    console.log('File successfully renamed to renamed_example.txt');
  }
});

// 5. Deleting a file (fs.unlink)
fs.unlink('renamed_example.txt', (err) => {
```

```
  if (err) {
    console.error('Error deleting the file:', err);
  } else {
    console.log('File successfully deleted!');
  }
});
```

program for demonstrating any 5 functions of console global object.

```
// 1. console.log() - Used to print standard output to the console
console.log('This is a standard log message.');

// 2. console.error() - Used to print error messages to the console
console.error('This is an error message!');

// 3. console.warn() - Used to print warning messages to the console
console.warn('This is a warning message!');

// 4. console.info() - Used to print informational messages to the console
console.info('This is an informational message.');

// 5. console.time() and console.timeEnd() - Used to measure time taken by a block of code
console.time('MyTimer');
for (let i = 0; i < 1000000; i++) {
  // Just an example loop to simulate some work
}
console.timeEnd('MyTimer'); // This will print the time taken for the loop execution
```

program for demonstrating any 5 functions of process global object.

```
// 1. process.hrtime() - Get high-resolution real time
const start = process.hrtime();
// Simulate some delay
setTimeout(() => {
  const end = process.hrtime(start);
  console.log(`High-resolution time taken: ${end[0]} seconds and ${end[1]} nanoseconds`);
}, 1000);

// 2. process.memoryUsage() - Memory usage of the process
const memoryUsage = process.memoryUsage();
console.log('Memory Usage (in bytes):', memoryUsage);

// 3. process.nextTick() - Deferred function execution
process.nextTick(() => {
  console.log('This is executed in the next event loop iteration!');
});

// 4. process.title - Get and set the process title
console.log('Current process title:', process.title);
process.title = 'MyCustomNodeApp';
console.log('Updated process title:', process.title);

// 5. process.on() - Listening for exit event
process.on('exit', (code) => {
```

```
  console.log(` Process is exiting with code ${code}` );
});

// Simulate a process exit (without calling process.exit())
setTimeout(() => {
  console.log('The program will exit now');
}, 2000);
```

program for demonstrating any 5 functions of OS utility module.

```
// Demonstrating 5 functions of Node.js 'fs' module for MERN Stack
const fs = require('fs');

function demonstrateFsModule() {
  console.log("Demonstrating 5 functions of the fs module:\n");

  // 1. Get the current working directory
  const cwd = process.cwd();
  console.log(` 1. Current Working Directory: ${cwd}\n` );

  // 2. Create a new directory
  const newDir = 'demo_dir';
  if (!fs.existsSync(newDir)) {
    fs.mkdirSync(newDir);
    console.log(` 2. Created a new directory: ${newDir}\n` );
  }
```

```
    // 3. List contents of the directory
    const contents = fs.readdirSync('.');
    console.log(` 3. Contents of the current directory: ${contents}\n `);

    // 4. Rename a file or directory
    const renamedDir = 'renamed_demo_dir';
    if (fs.existsSync(newDir)) {
        fs.renameSync(newDir, renamedDir);
        console.log(` 4. Renamed directory from ${newDir} to ${renamedDir}\n `);
    }

    // 5. Remove the directory
    if (fs.existsSync(renamedDir)) {
        fs.rmdirSync(renamedDir);
        console.log(` 5. Removed the directory: ${renamedDir}\n `);
    }
}

// Run the demonstration
demonstrateFsModule();
```

program for demonstrating any 5 functions of Path utility module.

```
// Demonstrating 5 functions of Node.js 'path' module for MERN Stack
const path = require('path');
```

```
function demonstratePathModule() {
    console.log("Demonstrating 5 functions of the path module:\n");

    // 1. Get the basename of a file path
    const filePath = '/user/local/bin/script.js';
    const baseName = path.basename(filePath);
    console.log(` 1. Basename of the file path '${filePath}': ${baseName}\n `);

    // 2. Get the directory name of a file path
    const dirName = path.dirname(filePath);
    console.log(` 2. Directory name of the file path '${filePath}': ${dirName}\n `);

    // 3. Get the file extension
    const fileExt = path.extname(filePath);
    console.log(` 3. File extension of '${filePath}': ${fileExt}\n `);

    // 4. Join multiple path segments
    const joinedPath = path.join('/user', 'local', 'bin', 'new_script.js');
    console.log(` 4. Joined path: ${joinedPath}\n `);

    // 5. Resolve a sequence of paths into an absolute path
    const absolutePath = path.resolve('src', 'scripts', 'app.js');
    console.log(` 5. Absolute path: ${absolutePath}\n `);
}

// Run the demonstration
```

```
demonstratePathModule();
```

5 functions of Node.js 'net' module for MERN Stack.

```
// Demonstrating 5 functions of Node.js 'net' module for MERN Stack
const net = require('net');

function demonstrateNetModule() {
  console.log("Demonstrating 5 functions of the net module:\n");

  // 1. Create a server
  const server = net.createServer((socket) => {
    console.log("1. Server created and client connected.");
    socket.write("Hello from server!\n");
    socket.end();
  });

  server.listen(8080, () => {
    console.log("   Server is listening on port 8080\n");
  });

  // 2. Get the address of the server
  server.on('listening', () => {
    const address = server.address();
    console.log(` 2. Server address: ${JSON.stringify(address)}\n `);
  });
```

```
  // 3. Create a client connection
  const client = net.createConnection({ port: 8080 }, () => {
    console.log("3. Client connected to server.\n");
  });

  // 4. Handle data from the server
  client.on('data', (data) => {
    console.log(` 4. Data received from server: ${data.toString()}\n ` );
  });

  // 5. Handle client disconnection
  client.on('end', () => {
    console.log("5. Client disconnected from server.\n");
    server.close(); // Close the server after the demonstration
  });
}

// Run the demonstration
demonstrateNetModule();
```

program for demonstrating any 3 functions of DNS utility module.

```
// Demonstrating 3 functions of Node.js 'dns' module for MERN Stack
const dns = require('dns');

function demonstrateDnsModule() {
  console.log("Demonstrating 3 functions of the dns module:\n");
```

```
    // 1. Resolve an address to an array of IP addresses
    dns.resolve('example.com', (err, addresses) => {
        if (err) {
            console.error(` 1. Error resolving addresses: ${err.message}` );
        } else {
            console.log(` 1. Resolved addresses for 'example.com': ${addresses}\n` );
        }
    });

    // 2. Perform a reverse lookup on an IP address
    const ip = '93.184.216.34'; // IP of example.com
    dns.reverse(ip, (err, hostnames) => {
        if (err) {
            console.error(` 2. Error performing reverse lookup: ${err.message}` );
        } else {
            console.log(` 2. Reverse lookup for IP '${ip}': ${hostnames}\n` );
        }
    });

    // 3. Get the DNS servers being used
    const servers = dns.getServers();
    console.log(` 3. Current DNS servers: ${servers}\n` );
}

// Run the demonstration
```

```
demonstrateDnsModule();
```

program for reading data from stream using Node.js.

```
// Demonstrating reading data from a stream using Node.js
const fs = require('fs');

function readDataFromStream() {
  console.log("Reading data from a stream:\n");

  // Create a readable stream from a file
  const filePath = 'example.txt'; // Ensure this file exists in the same directory
  const readableStream = fs.createReadStream(filePath, { encoding: 'utf8' });

  // Handle data event
  readableStream.on('data', (chunk) => {
    console.log(` Received chunk: \n${chunk}\n ` );
  });

  // Handle end event
  readableStream.on('end', () => {
    console.log("Finished reading the file.\n");
  });

  // Handle error event
  readableStream.on('error', (err) => {
    console.error(` Error reading the file: ${err.message}` );
```

```
  });
}

// Run the demonstration
readDataFromStream();
```

program for writing data to the stream using Node.js.

```
// Demonstrating writing data to a stream using Node.js
const fs = require('fs');

function writeDataToStream() {
  console.log("Writing data to a stream:\n");

  // Create a writable stream to a file
  const filePath = 'output.txt';
  const writableStream = fs.createWriteStream(filePath, { encoding: 'utf8' });

  // Write data to the stream
  writableStream.write("Hello, this is the first line.\n");
  writableStream.write("This is the second line.\n");

  // End the stream
  writableStream.end("This is the last line.\n");

  // Handle finish event
  writableStream.on('finish', () => {
```

```
    console.log("Finished writing to the file.\n");
  });

  // Handle error event
  writableStream.on('error', (err) => {
    console.error(`Error writing to the file: ${err.message}`);
  });
}

// Run the demonstration
writeDataToStream();
```

program for creating a module for arithmetic operations and use it in another program using Node.js.

```
// Arithmetic module: arithmetic.js
// This file contains functions for arithmetic operations
exports.add = (a, b) => a + b;
exports.subtract = (a, b) => a - b;
exports.multiply = (a, b) => a * b;
exports.divide = (a, b) => {
  if (b === 0) {
    throw new Error("Division by zero is not allowed.");
  }
  return a / b;
};

// Main program: app.js
```

```
// This file uses the arithmetic module for calculations
const arithmetic = require('./arithmetic');

function demonstrateArithmeticOperations() {
  console.log("Using the arithmetic module:\n");

  const a = 10;
  const b = 5;

  console.log(` Addition of ${a} and ${b}: ${arithmetic.add(a, b)}` );
  console.log(` Subtraction of ${a} and ${b}: ${arithmetic.subtract(a, b)}` );
  console.log(` Multiplication of ${a} and ${b}: ${arithmetic.multiply(a, b)}` );
  console.log(` Division of ${a} and ${b}: ${arithmetic.divide(a, b)}` );

  try {
    console.log(` Division of ${a} and 0: ${arithmetic.divide(a, 0)}` );
  } catch (error) {
    console.error(` Error: ${error.message}` );
  }
}

// Run the demonstration
demonstrateArithmeticOperations();
```

program for demonstrating any 5 functions of request object in Express.js.

```
// Demonstrating 5 functions of the request object in Express.js
```

```
const express = require('express');
const app = express();

// Middleware to parse JSON request bodies
app.use(express.json());

// 1. req.method: Display the HTTP method of the request
app.all('/method', (req, res) => {
  res.send(`HTTP Method: ${req.method}`);
});

// 2. req.url: Get the full URL of the request
app.get('/url', (req, res) => {
  res.send(`Requested URL: ${req.url}`);
});

// 3. req.headers: Access the headers sent with the request
app.get('/headers', (req, res) => {
  res.json({ headers: req.headers });
});

// 4. req.query: Access query parameters from the request
app.get('/query', (req, res) => {
  res.json({ queryParameters: req.query });
});
```

```
// 5. req.body: Access the body of the request (requires middleware)
app.post('/body', (req, res) => {
   res.json({ requestBody: req.body });
});

// Start the Express server
const PORT = 3000;
app.listen(PORT, () => {
   console.log(` Server is running on http://localhost:${PORT}` );
});
```

program for demonstrating any 5 functions of response object in Express.js.

```
const express = require('express');
const path = require('path');
const app = express();
const port = 3000;

// Route 1: Using res.send()
app.get('/send', (req, res) => {
  res.send('Hello from res.send()!');
});

// Route 2: Using res.json()
app.get('/json', (req, res) => {
  res.json({ message: 'Hello, this is a JSON response!' });
});
```

```
// Route 3: Using res.status()
app.get('/status', (req, res) => {
  res.status(404).send('Page not found');
});

// Route 4: Using res.sendFile()
app.get('/file', (req, res) => {
  // Serving a static HTML file
  const filePath = path.join(__dirname, 'index.html');
  res.sendFile(filePath);
});

// Route 5: Using res.redirect()
app.get('/redirect', (req, res) => {
  res.redirect('https://www.google.com');
});

app.listen(port, () => {
  console.log(` Server running at http://localhost:${port}` );
});
```

program for get HTTP method using Express.js.

```
const express = require('express');
const app = express();
const port = 3000;
```

```
// Define a GET route
app.get('/', (req, res) => {
  res.send('Hello, this is a response to a GET request!');
});

// Define another GET route with a dynamic parameter
app.get('/user/:name', (req, res) => {
  const userName = req.params.name;
  res.send(` Hello, ${userName}! This is your personalized GET response. ` );
});

// Define a GET route with query parameters
app.get('/search', (req, res) => {
  const { query } = req;
  res.json({
    message: 'Search results',
    query: query,
  });
});

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

program for post HTTP method using Express.js.

```
try {
  // Attempt to load required modules
  const express = require('express');
  const bodyParser = require('body-parser');

  console.log('Modules loaded successfully');

  // Initialize the app
  const app = express();
  const port = 3000;

  // Middleware to parse JSON bodies
  app.use(bodyParser.json());

  // Define a POST route
  app.post('/submit', (req, res) => {
    const { name, email } = req.body; // Extracting data from the request body
    if (name && email) {
      res.json({
        message: 'Data received successfully!',
        data: {
          name: name,
          email: email,
        },
      });
    } else {
```

```
    res.status(400).json({ error: 'Missing name or email in request body' });
   }
 });

 // Start the server
 app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
 });
} catch (err) {
 // Log any errors during module loading or initialization
 console.error('Error occurred:', err.message);
 console.error('Stack trace:', err.stack);
}
```

Using Postman:

Open Postman or any similar API testing tool.
Set the method to POST and the URL to http://localhost:3000/submit.
Go to the Body tab, select raw, and choose JSON as the format.
Enter the JSON body:

```
{
 "name": "John Doe",
 "email": "john.doe@example.com"
}
```

program for demonstrating the concept of Express.js router.

```
project-folder/
 ├── app.js
 ├── routes/
 │    ├── userRoutes.js
 │    └── productRoutes.js
```

App.js

```
// Import required modules
const express = require('express');

// Initialize the app
const app = express();
const port = 3000;

// Import routes
const userRoutes = require('./routes/userRoutes');
const productRoutes = require('./routes/productRoutes');

// Use routes
app.use('/users', userRoutes);    // Mount user-related routes under '/users'
app.use('/products', productRoutes); // Mount product-related routes under '/products'

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

```
});
```

userRoutes.js

```
// Import required modules
const express = require('express');
const router = express.Router();

// Mock data for demonstration
const users = [
 { id: 1, name: 'John Doe' },
 { id: 2, name: 'Jane Smith' }
];

// GET all users
router.get('/', (req, res) => {
 res.json(users);
});

// GET a single user by ID
router.get('/:id', (req, res) => {
 const userId = parseInt(req.params.id);
 const user = users.find(u => u.id === userId);

 if (user) {
  res.json(user);
```

```
  } else {
    res.status(404).json({ error: 'User not found' });
  }
});

// Export the router
module.exports = router;
```

productRoutes.js

```
// Import required modules
const express = require('express');
const router = express.Router();

// Mock data for demonstration
const products = [
  { id: 1, name: 'Laptop', price: 1000 },
  { id: 2, name: 'Smartphone', price: 500 }
];

// GET all products
router.get('/', (req, res) => {
  res.json(products);
});

// GET a single product by ID
router.get('/:id', (req, res) => {
```

```
  const productId = parseInt(req.params.id);
  const product = products.find(p => p.id === productId);

  if (product) {
    res.json(product);
  } else {
    res.status(404).json({ error: 'Product not found' });
  }
});

// Export the router
module.exports = router;
```

program for demonstrating the use of app.use() in Express.js.

```
// Import required modules
const express = require('express');

// Initialize the app
const app = express();
const port = 3000;

// Middleware function for logging each request
app.use((req, res, next) => {
  console.log(` [${new Date().toISOString()}] ${req.method} request to ${req.url}` );
  next(); // Pass control to the next middleware or route handler
});
```

```
// Middleware for parsing JSON request bodies
app.use(express.json());

// Example route 1: Home
app.get('/', (req, res) => {
  res.send('Welcome to the Home Page!');
});

// Example route 2: About
app.get('/about', (req, res) => {
  res.send('This is the About Page.');
});

// Middleware for handling 404 errors
app.use((req, res) => {
  res.status(404).send('Page not found.');
});

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

Create the MongoDB database and insert the records either using interface or using Node.js program

Open mongodb

Create a new connection and connect

Open mongo shell and write cmd:

Mongo

Create Database:

use myDatabase

Create collection and insert data:

```
db.users.insertMany([
 { name: "John Doe", age: 30, email: "john@example.com" },
 { name: "Jane Smith", age: 25, email: "jane@example.com" }
]);
```

Verify records:

```
db.users.find().pretty()
```

install mongodb in code terminal

app.js:

```
// Import MongoDB client
const { MongoClient } = require('mongodb');

// Connection URI
const uri = "mongodb://localhost:27017"; // Replace with your MongoDB URI if different

// Database and Collection Names
const dbName = "myDatabase";
const collectionName = "users";
```

```
// Data to Insert
const users = [
 { name: "John Doe", age: 30, email: "john@example.com" },
 { name: "Jane Smith", age: 25, email: "jane@example.com" }
];

// Main Function
async function main() {
 // Create a new MongoClient
 const client = new MongoClient(uri);

 try {
  // Connect to MongoDB
  await client.connect();
  console.log("Connected to MongoDB");

  // Get Database and Collection
  const db = client.db(dbName);
  const collection = db.collection(collectionName);

  // Insert Records
  const result = await collection.insertMany(users);
  console.log(` ${result.insertedCount} records inserted!`, result.insertedIds);

  // Fetch and Display Records
```

```
    const allUsers = await collection.find().toArray();
    console.log("All Records:", allUsers);
  } catch (error) {
    console.error("Error:", error);
  } finally {
    // Close the connection
    await client.close();
    console.log("Connection closed");
  }
}

// Execute the main function
main();
```

node app.js