Cd todo-list-mongodb

```

2. **Install Required Dependencies**:

Install the MongoDB Node.js driver to interact with MongoDB from your Node.js server:

```bash
Npm install mongodb
```

3. **Create a TaskList Component**:

Create a new file called `TaskList.js` in the `src` directory and define your `TaskList` component with the required functionality.

```jsx
Import React, { Component } from ‘react’;
Import ‘./TaskList.css’; // Create this CSS file for styling

Class TaskList extends Component {
 Constructor(props) {
  Super(props);
  This.state = {
   Tasks: [],
   newTask: ‘’,
  };
 }
```
```

```
  // Implement methods for adding, toggling completion, and filtering tasks

  Render() {
    // Render the task list, input field, filter buttons, and implement event handlers

    Return (
      <div className=”TaskList”>
        {/* Render the UI elements */}
      </div>
    );
  }
}

Export default TaskList;
```

4. **Implement MongoDB Integration**:

In the same component, add code to connect to MongoDB, insert and retrieve tasks from the “tasklist” collection, and handle state updates accordingly.

5. **Implement Task Management Logic**:

Inside your `TaskList` component, create methods to add new tasks, toggle task completion, and filter tasks based on their completion status.

6. **Styling**:

Create a CSS file (e.g., `TaskList.css`) to style your task list and related elements.

7. **Use the Component**:

Import and use the `TaskList` component in your `App.js` or any other parent component.

8. **Set Up MongoDB**:

Make sure you have MongoDB running locally or on a remote server. Create a database named “Task” and a collection named “tasklist” to store the tasks.

Note: In a production environment, you should set up a server to interact with MongoDB and not directly access the database from the React app for security reasons.

This is a high-level overview of how to create a basic To-Do list React component with MongoDB integration. The actual implementation involves coding the details of adding, toggling, filtering tasks, and handling the database interactions.

26-------

```
Import React, { Component } from ‘react’;
Import ‘./ExpenseTracker.css’; // Create this CSS file for styling

Class ExpenseTracker extends Component {
  Constructor() {
    Super();
    This.state = {
```

```
    Expenses: [],
    newExpense: {
     name: ‘’,
     amount: ‘’,
     category: ‘Food’,
     date: ‘’,
    },
    filterCategory: ‘All’,
    filterDate: ‘All’,
    error: null,
  };
 }

 // Implement methods for adding expenses, filtering, and handling errors

 Render() {
  Const { expenses, newExpense, filterCategory, filterDate, error } = this.state;

  Return (
   <div className=”ExpenseTracker”>
    {/* Render the list of expenses, input fields, filter controls, and error messages */}
   </div>
  );
 }
}

Export default ExpenseTracker
```

27-----

Creating a React component for a weather application with the specified features involves setting up the component, handling user input, fetching weather data, and displaying it. Here's a simplified example:

```jsx
Import React, { Component } from 'react';
Import './WeatherApp.css'; // Create this CSS file for styling

Class WeatherApp extends Component {
  Constructor() {
    Super();
    This.state = {
      cityName: '',
      weatherData: null,
      error: null,
    };
  }

  handleCityChange = (event) => {
    this.setState({ cityName: event.target.value });
  };

  fetchWeatherData = () => {
    const apiKey = 'YOUR_API_KEY'; // Replace with your actual API key
    const { cityName } = this.state;
    const apiUrl =
`https://api.openweathermap.org/data/2.5/weather?q=${cityName}&appid=${apiKey}&units=metric`;
```

```
  fetch(apiUrl)
    .then((response) => response.json())
    .then((data) => {
      If (data.cod === 200) {
        This.setState({ weatherData: data, error: null });
      } else {
        This.setState({ weatherData: null, error: data.message });
      }
    })
    .catch((error) => {
      This.setState({ weatherData: null, error: ‘Network issue. Please try again.’ });
    });
};

Render() {
  Const { cityName, weatherData, error } = this.state;

  Return (
    <div className=”WeatherApp”>
      <h2>Weather App</h2>
      <input
        Type=”text”
        Placeholder=”Enter city name”
        Value={cityName}
        onChange={this.handleCityChange}
      />
      <button onClick={this.fetchWeatherData}>Get Weather</button>

      {error && <p className=”error”>{error}</p>}
```

```
      {weatherData && (
        <div>
          <h3>{weatherData.name}</h3>
          <p>Temperature: {weatherData.main.temp}°C</p>
          <p>Weather: {weatherData.weather[0].description}</p>
        </div>
      )}
    </div>
  );
 }
}

Export default WeatherApp;
```

In this example:

1. We have an input field for entering the city name, a button to fetch weather data, and a display area for the weather information.

2. We use the OpenWeatherMap API (replace 'YOUR_API_KEY' with your actual API key) to fetch weather data based on the user's input.

3. We handle errors, such as a city not found or network issues, by displaying error messages.

4. When successful, we display the city name, temperature, and weather description.

Remember to replace 'YOUR_API_KEY' with a valid API key from OpenWeatherMap or any other weather data provider you prefer. Additionally, you can style the component by creating a CSS file (`WeatherApp.css`) to make it visually appealing.

28---

Creating a ReactJS script for a form with the specified fields and inserting the submitted values into a MongoDB database requires a combination of frontend and backend development. Below is an example of how you can create the frontend component in React and outline the backend setup.

Frontend (ReactJS):

1. Create a React component that represents the form with the specified fields and handles user input:

```jsx
Import React, { Component } from 'react';

Class UserForm extends Component {
 Constructor() {
  Super();
  This.state = {
   City: 'Ahmedabad',
   bloodGroup: 'O+',
  };
 }

 handleCityChange = (event) => {
  this.setState({ city: event.target.value });
 };

 handleBloodGroupChange = (event) => {
  this.setState({ bloodGroup: event.target.value });
 };

 handleSubmit = () => {
```

```
  const { city, bloodGroup } = this.state;

  // Send the data to the backend for database insertion
  Fetch('/api/addUser', {
    Method: 'POST',
    Headers: {
      'Content-Type': 'application/json',
    },
    Body: JSON.stringify({ city, bloodGroup }),
  })
    .then((response) => response.json())
    .then((data) => {
      Console.log(data);
      // Reset the form or show a success message as needed
    })
    .catch((error) => {
      Console.error('Error:', error);
    });
};

Render() {
  Return (
    <div>
      <h2>User Registration Form</h2>
      <div>
        <label>
          City:
          <select value={this.state.city} onChange={this.handleCityChange}>
            <option value="Ahmedabad">Ahmedabad</option>
```

```
            <option value=”Rajkot”>Rajkot</option>
            <option value=”Surat”>Surat</option>
            <option value=”Vadodara”>Vadodara</option>
          </select>
        </label>
      </div>
      <div>
        <label>
          Blood Group:
          <input
            Type=”radio”
            Name=”bloodGroup”
            Value=”O+”
            Checked={this.state.bloodGroup === ‘O+’}
            onChange={this.handleBloodGroupChange}
          />
          O+
          {/* Repeat similar input elements for other blood group options */}
        </label>
      </div>
      <button onClick={this.handleSubmit}>Submit</button>
    </div>
  );
 }
}

Export default UserForm;
```

Backend (Node.js with Express and MongoDB):

Set up a backend server using Node.js and Express to handle the POST request for adding a user to the MongoDB database. Here's a basic outline:

1. Install the required dependencies (Express, Mongoose for MongoDB, etc.):

```bash
Npm install express mongoose body-parser
```

2. Create a server file (e.g., `server.js`) for your Node.js backend:

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

Const app = express();
Const port = process.env.PORT || 5000;

// Connect to MongoDB (replace 'mongodb://localhost/mydb' with your MongoDB connection URL)
Mongoose.connect('mongodb://localhost/LJU', { useNewUrlParser: true, useUnifiedTopology: true });
Const db = mongoose.connection;

Db.on('error', console.error.bind(console, 'MongoDB connection error:'));
Db.once('open', () => {
  Console.log('Connected to MongoDB');
});
```

```
// Define a User schema and model (adjust as needed)
Const userSchema = new mongoose.Schema({
  City: String,
  bloodGroup: String,
});

Const User = mongoose.model('User', userSchema);

// Middleware for parsing JSON requests
App.use(bodyParser.json());

// Define a route to add a user to the database
App.post('/api/addUser', (req, res) => {
  Const { city, bloodGroup } = req.body;

  Const newUser = new User({ city, bloodGroup });

  newUser.save((err) => {
    if (err) {
      console.error('Error inserting user:', err);
      res.status(500).json({ error: 'Error inserting user' });
    } else {
      Console.log('User inserted successfully');
      Res.json({ message: 'User inserted successfully' });
    }
  });
});
```

```
// Start the server
App.listen(port, () => {
  Console.log(`Server is running on port ${port}`);
});
```

3. Replace the MongoDB connection URL with your actual MongoDB connection URL.

4. Run your Node.js server using `node server.js`.

This setup provides a basic structure for handling the frontend form submission and inserting the submitted data into a MongoDB database. You'll need to replace the connection URL and adapt the code as needed for your specific application

29----

```
Import React from 'react';
Import { BrowserRouter as Router, Route, Link, Switch, Redirect } from 'react-router-dom';

Const SubjectIndex = () => (
  <div>
    <h2>Subject Index</h2>
    <ul>
      <li>
        <Link to="/fsd2/json">JSON</Link>
      </li>
      <li>
        <Link to="/fsd2/nodejs">NodeJS/ExpressJS</Link>
      </li>
```

```
    <li>
      <Link to=”/fsd2/reactjs”>React JS</Link>
    </li>
   </ul>
  </div>
);

Const Content = () => (
  <div>
   <h2>Content</h2>
   <table>
    <tr>
     <th>Topic</th>
     <th>Details</th>
    </tr>
    <tr>
     <td>JSON</td>
     <td>JSON content goes here.</td>
    </tr>
    <tr>
     <td>NodeJS/ExpressJS</td>
     <td>NodeJS/ExpressJS content goes here.</td>
    </tr>
    <tr>
     <td>React JS</td>
     <td>React JS content goes here.</td>
    </tr>
   </table>
  </div>
```

```
);

Const NoPage = () => (
  <div>
    <h2>No page found</h2>
    <p>Sorry, the page you requested does not exist.</p>
  </div>
);

Const App = () => (
  <Router>
    <div>
      <Switch>
        <Route path="/fsd2" exact component={SubjectIndex} />
        <Route path="/fsd2/json" component={Content} />
        <Route path="/fsd2/nodejs" component={Content} />
        <Route path="/fsd2/reactjs" component={Content} />
        <Route path="/no-page" component={NoPage} />
        <Redirect from="/" to="/fsd2" />
        <Redirect to="/no-page" />
      </Switch>
    </div>
  </Router>
);

Export default App;
```

Creating a ReactJS script for a form with two fields (Subject Name and Marks) and inserting the entered values into a MongoDB database requires a combination of frontend and backend development. Below is an example of how you can create the frontend component in React and outline the backend setup.

Frontend (ReactJS):

1. Create a React component that represents the form with the specified fields and handles user input:

```jsx
Import React, { Component } from 'react';

Class StudentForm extends Component {
 Constructor() {
  Super();
  This.state = {
   subjectName: 'FSD2',
   marks: '',
  };
 }

 handleSubjectNameChange = (event) => {
  this.setState({ subjectName: event.target.value });
 };

 handleMarksChange = (event) => {
  this.setState({ marks: event.target.value });
 };
```

```
handleSubmit = () => {
  const { subjectName, marks } = this.state;

  // Send the data to the backend for database insertion
  Fetch('/api/addStudentData', {
    Method: 'POST',
    Headers: {
      'Content-Type': 'application/json',
    },
    Body: JSON.stringify({ subjectName, marks }),
  })
    .then((response) => response.json())
    .then((data) => {
      Console.log(data);
      // Reset the form or show a success message as needed
    })
    .catch((error) => {
      Console.error('Error:', error);
    });
};

Render() {
  Return (
    <div>
      <h2>Student Data Form</h2>
      <div>
        <label>
          Subject Name:
```

```
          <select value={this.state.subjectName} onChange={this.handleSubjectNameChange}>
            <option value=”FSD2”>FSD2</option>
            <option value=”FCSP2”>FCSP2</option>
            <option value=”DS”>DS</option>
            <option value=”TOC”>TOC</option>
            <option value=”COA”>COA</option>
          </select>
        </label>
      </div>
      <div>
        <label>
          Marks:
          <input
            Type=”text”
            Value={this.state.marks}
            onChange={this.handleMarksChange}
          />
        </label>
      </div>
      <button onClick={this.handleSubmit}>Submit</button>
    </div>
  );
 }
}

Export default StudentForm;
```

Backend (Node.js with Express and MongoDB):

Set up a backend server using Node.js and Express to handle the POST request for adding student data to the MongoDB database. Here's a basic outline:

1. Install the required dependencies (Express, Mongoose for MongoDB, etc.):

```bash
Npm install express mongoose body-parser
```

2. Create a server file (e.g., `server.js`) for your Node.js backend:

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

Const app = express();
Const port = process.env.PORT || 5000;

// Connect to MongoDB (replace 'mongodb://localhost/mydb' with your MongoDB connection URL)
Mongoose.connect('mongodb://localhost/StudentData', { useNewUrlParser: true, useUnifiedTopology: true });
Const db = mongoose.connection;

Db.on('error', console.error.bind(console, 'MongoDB connection error:'));
Db.once('open', () => {
  Console.log('Connected to MongoDB');
});
```

```
// Define a StudentData schema and model (adjust as needed)
Const studentDataSchema = new mongoose.Schema({
  subjectName: String,
  marks: Number,
});

Const StudentData = mongoose.model('Information', studentDataSchema);

// Middleware for parsing JSON requests
App.use(bodyParser.json());

// Define a route to add student data to the database
App.post('/api/addStudentData', (req, res) => {
  Const { subjectName, marks } = req.body;

  Const newStudentData = new StudentData({ subjectName, marks });

  newStudentData.save((err) => {
    if (err) {
      console.error('Error inserting student data:', err);
      res.status(500).json({ error: 'Error inserting student data' });
    } else {
      Console.log('Student data inserted successfully');
      Res.json({ message: 'Student data inserted successfully' });
    }
  });
});
```

```
// Start the server
App.listen(port, () => {
  Console.log(`Server is running on port ${port}`);
});
```

3. Replace the MongoDB connection URL with your actual MongoDB connection URL.

4. Run your Node.js server using `node server.js`.

This setup provides a basic structure for handling the frontend form submission and inserting the submitted data into a MongoDB database. You'll need to adapt the code as needed for your specific application and replace the connection URL with your actual MongoDB connection URL.

31----

To create an HTML form for collecting data and apply validations, you can use HTML, JavaScript, and MongoDB. Here's an example of an HTML form with validation rules and the corresponding Node.js backend to insert data into a MongoDB collection:

****HTML Form (form.html):****

```
```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Exam Registration</title>
```
```

```
</head>
<body>
  <h1>Exam Registration</h1>
  <form id="examForm">
    <label for="name">Name (Max 12 characters, uppercase only):</label>
    <input type="text" id="name" name="name" maxlength="12" pattern="[A-Z]+" required><br><br>

    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required><br><br>

    <label for="city">City (Ahmedabad, Gandhinagar, Vadodara):</label>
    <input type="text" id="city" name="city" pattern="^(Ahmedabad|Gandhinagar|Vadodara)$" required><br><br>

    <label for="examDate">Date of Exam (Between 1-10-2023 and 12-10-2023):</label>
    <input type="date" id="examDate" name="examDate" min="2023-10-01" max="2023-10-12" required><br><br>

    <input type="submit" value="Submit">
  </form>

  <script src="form.js"></script>
</body>
</html>
```

JavaScript for Form Validation (form.js):

```javascript
```

```
Document.addEventListener("DOMContentLoaded", function () {
  Const form = document.getElementById("examForm");

  Form.addEventListener("submit", function (event) {
    Event.preventDefault();

    // Get form values
    Const name = document.getElementById("name").value.trim().toUpperCase();
    Const email = document.getElementById("email").value;
    Const city = document.getElementById("city").value.toLowerCase();
    Const examDate = document.getElementById("examDate").value;

    // Perform additional validation here if needed
    // …

    // Create an object with the form data
    Const formData = {
      Name,
      Email,
      City,
      examDate,
    };

    // Send the data to the server for insertion into MongoDB
    Fetch("/api/addExamData", {
      Method: "POST",
      Headers: {
        "Content-Type": "application/json",
      },
```

```
    Body: JSON.stringify(formData),
  })
    .then((response) => response.json())
    .then((data) => {
      Console.log(data);
      // Handle success or display error messages to the user
    })
    .catch((error) => {
      Console.error("Error:", error);
      // Handle errors or display error messages to the user
    });
 });
});
```

Node.js Backend for MongoDB Insertion (server.js):

Assuming you have MongoDB set up and running, you can create a Node.js server to handle the form submission and insert data into the MongoDB collection. You'll need to use a MongoDB driver like Mongoose. Here's a simplified example:

```javascript
Const express = require("express");
Const mongoose = require("mongoose");
Const bodyParser = require("body-parser");

Const app = express();
Const port = process.env.PORT || 3000;
```