Ing. Luis Guillermo Molero Suárez

Aplicación CRUD de Stack MERN React

En este ejercicio se creará una aplicación básica para estudiantes. Esta aplicación permitirá crear estudiantes, mostrar la lista de estudiantes, actualizarlos y eliminarlos de la base de datos de MongoDB.

Configuración:

- Configuración del proyecto React
- Creando componentes en React
- Construir y trabajar con routers React
- Trabajar con React-Bootstrap
- Introducción a los formularios React
- Consumir API REST en la aplicación React
- Comenzando con Crear, Leer, Actualizar y Eliminar en React
- Configuración de un servidor Node y Express
- MongoDB configurado en el proyecto de pila MERN
- Creación de API REST con Express.js
- Realización de solicitudes HTTP con la biblioteca React Axios

Prerrequisitos

Antes de comenzar con este ejercicio, se debe conocer los fundamentos de React.js y HTML, CSS, JavaScript, TypeScript o ES6. Consulte el sitio web oficial de React para obtener más información sobre sus funciones, conceptos centrales y referencias.

https://reactjs.org/docs/react-api.html

Para construir la aplicación web MERN Stack, debe tener Node.js instalado en su sistema. Consulte el siguiente sitio web para mayor información:

https://nodejs.org/dist/latest-v14.x/docs/api/documentation.html

Una vez que el Node.Js esté instalado, ejecute en cmd el siguiente comando para verificar la versión de Node.js:

```
node -v
```

Crear aplicación React

Comencemos a construir el proyecto React con create-react-app (CRA).

```
npx create-react-app react-mernstack-crud
```

Ingrese a la carpeta del proyecto React:

```
cd react-mernstack-crud
```

Para iniciar el proyecto React MERN Stack, ejecute el siguiente comando:

```
yarn start
```

Este comando abre el proyecto React en la siguiente URL:

http://localhost:3000/

Integración de React Bootstrap con la aplicación React

En el siguiente paso, se instalará el framework front-end React Bootstrap en nuestra aplicación del Stack MERN. Este framework nos permitirá usar el componente UI de Bootstrap en nuestra aplicación React CRUD.
React Bootstrap, nos permite importar componentes individuales de la interfaz de usuario en lugar de importar todo el conjunto de bibliotecas.

```
npm install react-bootstrap Bootstrap
```

Tienes que importar Bootstrap en el archivo `src/App.js` y te ayudará a crear los componentes de la interfaz de usuario rápidamente.

```
import "bootstrap/dist/css/bootstrap.css";
```

Creación de componentes simples de React

En este paso, aprenderemos a crear componentes de React para administrar datos en la aplicación CRUD. Para ello, dirígete a la carpeta `src`, crea una carpeta y asígnale el nombre de `components` y dentro de ese directorio crea los siguientes componentes.

```
create-student.component.js
edit-student.component.js
student-list.component.js
```

Vaya a `src/components/create-student.component.js` y agregue el siguiente código:

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

export default class CreateStudent extends Component {
  render() {
    return (
      <div>
        <p>React Create Student Component!</p>
      </div>
    );
  }
}
```

Vaya a `src/components/edit-student.component.js` y agregue el siguiente código:

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

export default class EditStudent extends Component {
  render() {
    return (
      <div>
        <p>React Edit Student Component!</p>
      </div>
    );
  }
}
```

Vaya a `src/components/student-list.component.js` y agregue el siguiente código:

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

export default class StudentList extends Component {
  render() {
    return (
      <div>
        <p>React Student List Component!</p>
      </div>
    );
  }
}
```

Implementando React Router

En este paso, implementaremos Router en la aplicación React.js. Ingrese el siguiente comando en cmd:

```
npm install react-router-dom --save
```

Debe crear el archivo `src/serviceWorker.js`. A continuación, agregue el siguiente código dentro del archivo:

```
// This optional code is used to register a service worker.
// register() is not called by default.

// This lets the app load faster on subsequent visits in production, and
gives
// it offline capabilities. However, it also means that developers (and u
sers)
// will only see deployed updates on subsequent visits to a page, after a
ll the
// existing tabs open on the page have been closed, since previously cach
ed
// resources are updated in the background.

// To learn more about the benefits of this model and instructions on how
 to
// opt-in, read https://bit.ly/CRA-PWA

const isLocalhost = Boolean(
  window.location.hostname === 'localhost' ||
    // [::1] is the IPv6 localhost address.
    window.location.hostname === '[::1]' ||
    // 127.0.0.1/8 is considered localhost for IPv4.
    window.location.hostname.match(
```

```
      /^127(?:\.(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)){3}$/
    )
);

export function register(config) {
  if (process.env.NODE_ENV === 'production' && 'serviceWorker' in navigat
or) {
    // The URL constructor is available in all browsers that support SW.
    const publicUrl = new URL(process.env.PUBLIC_URL, window.location.hre
f);
    if (publicUrl.origin !== window.location.origin) {
      // Our service worker won't work if PUBLIC_URL is on a different or
igin
      // from what our page is served on. This might happen if a CDN is u
sed to
      // serve assets; see https://github.com/facebook/create-react-
app/issues/2374
      return;
    }

    window.addEventListener('load', () => {
      const swUrl = `${process.env.PUBLIC_URL}/service-worker.js`;

      if (isLocalhost) {
        // This is running on localhost. Let's check if a service worker
still exists or not.
        checkValidServiceWorker(swUrl, config);

        // Add some additional logging to localhost, pointing developers
to the
        // service worker/PWA documentation.
        navigator.serviceWorker.ready.then(() => {
          console.log(
            'This web app is being served cache-first by a service ' +
              'worker. To learn more, visit https://bit.ly/CRA-PWA'
          );
        });
      } else {
        // Is not localhost. Just register service worker
        registerValidSW(swUrl, config);
      }
    });
  }
}

function registerValidSW(swUrl, config) {
  navigator.serviceWorker
    .register(swUrl)
    .then(registration => {
      registration.onupdatefound = () => {
```

```
        const installingWorker = registration.installing;
        if (installingWorker == null) {
          return;
        }
        installingWorker.onstatechange = () => {
          if (installingWorker.state === 'installed') {
            if (navigator.serviceWorker.controller) {
              // At this point, the updated precached content has been fe
tched,
              // but the previous service worker will still serve the old
er
              // content until all client tabs are closed.
              console.log(
                'New content is available and will be used when all ' +
                  'tabs for this page are closed. See https://bit.ly/CRA-
PWA.'
              );

              // Execute callback
              if (config && config.onUpdate) {
                config.onUpdate(registration);
              }
            } else {
              // At this point, everything has been precached.
              // It's the perfect time to display a
              // "Content is cached for offline use." message.
              console.log('Content is cached for offline use.');

              // Execute callback
              if (config && config.onSuccess) {
                config.onSuccess(registration);
              }
            }
          }
        };
      };
    })
    .catch(error => {
      console.error('Error during service worker registration:', error);
    });
}

function checkValidServiceWorker(swUrl, config) {
  // Check if the service worker can be found. If it can't reload the pag
e.
  fetch(swUrl)
    .then(response => {
      // Ensure service worker exists, and that we really are getting a J
S file.
      const contentType = response.headers.get('content-type');
```

```
      if (
        response.status === 404 ||
        (contentType != null && contentType.indexOf('javascript') === -1)
      ) {
        // No service worker found. Probably a different app. Reload the
page.
        navigator.serviceWorker.ready.then(registration => {
          registration.unregister().then(() => {
            window.location.reload();
          });
        });
      } else {
        // Service worker found. Proceed as normal.
        registerValidSW(swUrl, config);
      }
    })
    .catch(() => {
      console.log(
        'No internet connection found. App is running in offline mode.'
      );
    });
}

export function unregister() {
  if ('serviceWorker' in navigator) {
    navigator.serviceWorker.ready.then(registration => {
      registration.unregister();
    });
  }
}
```

Consulte el siguiente sitio web oficial para mayor información:

https://developer.mozilla.org/es/docs/Web/API/Service_Worker_API

A continuación, diríjase al archivo `src/index.js` y vincule el componente de la aplicación con la ayuda del objeto `<BrowserRouter>`.

```
import React from "react";
import ReactDOM from "react-dom";
import { BrowserRouter } from "react-router-dom";

import "./index.css";
import App from "./App";
import * as serviceWorker from "./serviceWorker";

ReactDOM.render(
```

```
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  document.getElementById("root")
);

// If you want your app to work offline and load faster, you can c
hange
// unregister() to register() below. Note this comes with some pit
falls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

A continuación, incluya el menú en nuestra aplicación React CRUD. Agregue el código a continuación en `src/App.js`:

```
import React from "react";
import Nav from "react-bootstrap/Nav";
import Navbar from "react-bootstrap/Navbar";
import Container from "react-bootstrap/Container";
import Row from "react-bootstrap/Row";
import Col from "react-bootstrap/Col";
import "bootstrap/dist/css/bootstrap.css";
import "./App.css";

import { BrowserRouter as Router, Switch, Route, Link } from "reac
t-router-dom";

import CreateStudent from "./components/create-student.component";
import EditStudent from "./components/edit-student.component";
import StudentList from "./components/student-list.component";

function App() {
  return (<Router>
    <div className="App">
      <header className="App-header">
        <Navbar bg="dark" variant="dark">
          <Container>

            <Navbar.Brand>
              <Link to={"/create-student"} className="nav-link">
                React MERN Stack App
              </Link>
            </Navbar.Brand>

            <Nav className="justify-content-end">
              <Nav>
                <Link to={"/create-student"} className="nav-link">
                  Create Student
                </Link>
              </Nav>

              {/* <Nav>
                <Link to={"/edit-student/:id"} className="nav-
link">
                  Edit Student
                </Link>
              </Nav> */}
```

```
              <Nav>
                <Link to={"/student-list"} className="nav-link">
                  Student List
                </Link>
              </Nav>
            </Nav>

          </Container>
        </Navbar>
      </header>

      <Container>
        <Row>
          <Col md={12}>
            <div className="wrapper">
              <Switch>
                <Route exact path='/' component={CreateStudent} />
                <Route path="/create-
student" component={CreateStudent} />
                <Route path="/edit-
student/:id" component={EditStudent} />
                <Route path="/student-
list" component={StudentList} />
              </Switch>
            </div>
          </Col>
        </Row>
      </Container>
    </div>
  </Router>);
}

export default App;
```

Si presenta el siguiente ERROR, `[HMR] Waiting for update signal from WDS...`
LA SOLUCIÓN:

https://stackoverflow.com/questions/59695102/reactjs-console-error-hmr-waiting-for-update-signal-from-wds

https://github.com/facebook/create-react-app/issues/8153

Una vez solucionado, cerramos nuestra web y reiniciamos con el siguiente comando:

```
yarn start
```

Con esto, hemos finalizado nuestra interfaz inicial y su navegabilidad.
Lo siguiente, será diseñar la lógica de nuestra SPA

Lógica de nuestro CRUD en React Js

Crear formulario React con React Bootstrap

En este paso, crearemos el formulario utilizando el marco front-end React Bootstrap para enviar los datos del estudiante en el componente `create-student.component.js`.

```
import React, {Component} from "react";
import Form from 'react-bootstrap/Form'
import Button from 'react-bootstrap/Button';

export default class CreateStudent extends Component {
  render() {
    return (<div class="form-wrapper">
      <Form>
        <Form.Group controlId="Name">
          <Form.Label>Name</Form.Label>
          <Form.Control type="text"/>
        </Form.Group>

        <Form.Group controlId="Email">
          <Form.Label>Email</Form.Label>
          <Form.Control type="email"/>
        </Form.Group>

        <Form.Group controlId="Name">
          <Form.Label>Roll No</Form.Label>
          <Form.Control type="text"/>
        </Form.Group>

        <Button variant="danger" size="lg" block="block" type="sub
mit">
          Create Student
        </Button>
      </Form>
    </div>);
  }
}
```

Enviar datos de formularios en React

A continuación, aprenderemos a enviar los datos de Formularios en React.js. Ya hemos creado el formulario del estudiante y debemos enviar el nombre, el correo electrónico y el número de lista del estudiante a la base de datos.

Comenzaremos creando el constructor dentro de la clase del componente créate-student. Luego, establezca el estado inicial del componente CreateStudent estableciendo this.state Object.

Luego declare las diversas funciones con cada valor de campo del formulario de React, de modo que cuando el usuario inserte los datos dentro del campo de entrada del formulario, se establecerá un estado en consecuencia.

A continuación, debemos definir el evento de envío, que nos permitirá crear nuevos datos de los estudiantes cuando el usuario haga clic en el botón de envío "Crear estudiante". Copiamos el siguiente código en el archivo create-student.component.js:

```
import React, {Component} from "react";
import Form from 'react-bootstrap/Form'
import Button from 'react-bootstrap/Button';

export default class CreateStudent extends Component {

  constructor(props) {
    super(props)

    // Setting up functions
    this.onChangeStudentName = this.onChangeStudentName.bind(this)
;
    this.onChangeStudentEmail = this.onChangeStudentEmail.bind(thi
s);
    this.onChangeStudentRollno = this.onChangeStudentRollno.bind(t
his);
    this.onSubmit = this.onSubmit.bind(this);

    // Setting up state
    this.state = {
      name: '',
      email: '',
      rollno: ''
    }
  }

  onChangeStudentName(e) {
    this.setState({name: e.target.value})
  }

  onChangeStudentEmail(e) {
```

```
    this.setState({email: e.target.value})
  }

  onChangeStudentRollno(e) {
    this.setState({rollno: e.target.value})
  }

  onSubmit(e) {
    e.preventDefault()

    console.log(`Student successfully created!`);
    console.log(`Name: ${this.state.name}`);
    console.log(`Email: ${this.state.email}`);
    console.log(`Roll no: ${this.state.rollno}`);

    this.setState({
      name: '',
      email: '',
      rollno: ''
    });

  }

  render() {
    return (<div className="form-wrapper">
      <Form onSubmit={this.onSubmit}>
        <Form.Group controlId="Name">
          <Form.Label>Name</Form.Label>
          <Form.Control type="text" value={this.state.name} onChan
ge={this.onChangeStudentName}/>
        </Form.Group>

        <Form.Group controlId="Email">
          <Form.Label>Email</Form.Label>
          <Form.Control type="email" value={this.state.email} onCh
ange={this.onChangeStudentEmail}/>
        </Form.Group>

        <Form.Group controlId="Name">
          <Form.Label>Roll No</Form.Label>
          <Form.Control type="text" value={this.state.rollno} onCh
ange={this.onChangeStudentRollno}/>
        </Form.Group>

        <Button variant="danger" size="lg" block="block" type="sub
mit">
          Create Student
```

```
          </Button>
        </Form>
      </div>);
  }
}
```

Con esto, hemos finalizado nuestra lógica del CRUD en Reack Js.

Lo siguiente, será diseñar el Backend de nuestra SPA

Compilación del NodeJs Backend para nuestro MERN Stack

Crearemos una carpeta dentro de nuestra aplicación React para administrar los servicios "backend" como base de datos, modelos, esquema, rutas y API.

Creemos la siguiente carpeta: `react-mernstack-crud/backend`.

Luego, necesitamos crear un archivo `package.json` en la carpeta: `react-mernstack-crud/backend` separado para administrar el backend de nuestra SPA. Para ello, ejecutamos el siguiente comando desde cms y damos ENTER a todas las preguntas que nos hace la interfaz:

```
npm init
```

A continuación, instale las dependencias de nodo que se indican a continuación para el backend:

```
npm install mongoose express cors body-parser
```

Instale la dependencia de nodemon para automatizar el proceso de reinicio del servidor:

```
npm install nodemon --save-dev
```

Nuestro archivo final `package.json` se verá de la siguiente forma:

Asegúrese de actualizar la propiedad "`main`" cambiando el nombre de "`index.js`" a "`server.js`".

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.19.0",
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "mongoose": "^5.13.0"
  },
  "devDependencies": {
```

```
    "nodemon": "^2.0.9"
  }
}
```

Con esto, hemos finalizado nuestro Backend del CRUD en Reack Js.

Lo siguiente, será diseñar la persistencia de nuestra SPA

Persistencia

Configuración de la base de datos MongoDB

A continuación, configuraremos una base de datos MongoDB para la aplicación React MERN. Ya hemos instalado mongoDB, ahora, creamos la carpeta `database` dentro de la carpeta de `backend` y crearemos un archivo con el nombre de `db.js` y pegamos el siguiente código:

```
module.exports = {
    db: 'mongodb://localhost:27017/reactdb'
};
```

Hemos declarado la base de datos mongoDB llamada “reactdb”. A nivel local, no requiere nombre de usuario y contraseña; sin embargo, en la producción, debe crear un administrador y asignar la base de datos a un usuario específico.

Definir esquema de mangosta

Luego, crea un esquema mongoDB para interactuar con la base de datos mongoDB. Cree la carpeta `react-mernstack-crud/backend/models` para mantener los archivos relacionados con el esquema y cree un archivo `Student.js` dentro de ella.

A continuación, incluya el siguiente código en el archivo `backend/models/Student.js`:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

let studentSchema = new Schema({
  name: {
    type: String
  },
  email: {
    type: String
  },
  rollno: {
    type: Number
  }
}, {
    collection: 'students'
  })

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

Declaramos un “nombre”, “correo electrónico” y campos “rollno” junto con sus respectivos tipos de datos en el esquema del estudiante.

Crear rutas usando Express / Node JS para la aplicación React CRUD

En este paso, estamos construyendo las rutas (API REST) para la aplicación React CRUD CREATE, READ, UPDATE y DELETE usando Express y Node.js. Estas rutas nos ayudarán a administrar los datos en nuestra aplicación para estudiantes React MERN.

Crea la siguiente carpeta `react-mernstack-crud/backend/routes`, aquí guardaremos todos los archivos relacionados con las rutas. Además, cree el archivo `student.route.js` dentro de esta carpeta, en este archivo definiremos las API REST.

Luego, vaya al archivo `react-mernstack-crud/backend/routes/student.route.js` y agregue el siguiente código:

```
let mongoose = require('mongoose'),
  express = require('express'),
  router = express.Router();

// Student Model
let studentSchema = require('../models/Student');

// CREATE Student
router.route('/create-student').post((req, res, next) => {
  studentSchema.create(req.body, (error, data) => {
    if (error) {
      return next(error)
    } else {
      console.log(data)
      res.json(data)
    }
  })
});

// READ Students
router.route('/').get((req, res) => {
  studentSchema.find((error, data) => {
    if (error) {
      return next(error)
    } else {
      res.json(data)
    }
  })
})
```

```
// Get Single Student
router.route('/edit-student/:id').get((req, res) => {
  studentSchema.findById(req.params.id, (error, data) => {
    if (error) {
      return next(error)
    } else {
      res.json(data)
    }
  })
})


// Update Student
router.route('/update-student/:id').put((req, res, next) => {
  studentSchema.findByIdAndUpdate(req.params.id, {
    $set: req.body
  }, (error, data) => {
    if (error) {
      return next(error);
      console.log(error)
    } else {
      res.json(data)
      console.log('Student updated successfully !')
    }
  })
})

// Delete Student
router.route('/delete-student/:id').delete((req, res, next) => {
  studentSchema.findByIdAndRemove(req.params.id, (error, data) =>
{
    if (error) {
      return next(error);
    } else {
      res.status(200).json({
        msg: data
      })
    }
  })
})

module.exports = router;
```

Configurar Server.js en Node / Express.js Backend

Casi hemos creado todo para configurar el backend de Node y Expresss.js para la aplicación React CRUD. Ahora crearemos el archivo server.js en la siguiente dirección: `react-mernstack-crud/backend/`

Pegue el siguiente código dentro del archivo `backend/server.js`:

```
let express = require('express');
let mongoose = require('mongoose');
let cors = require('cors');
let bodyParser = require('body-parser');
let dbConfig = require('./database/db');

// Express Route
const studentRoute = require('../backend/routes/student.route')

// Connecting mongoDB Database
mongoose.Promise = global.Promise;
mongoose.connect(dbConfig.db, {
  useNewUrlParser: true
}).then(() => {
  console.log('Database sucessfully connected!')
},
  error => {
    console.log('Could not connect to database : ' + error)
  }
)

const app = express();
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
  extended: true
}));
app.use(cors());
app.use('/students', studentRoute)


// PORT
const port = process.env.PORT || 4000;
const server = app.listen(port, () => {
  console.log('Connected to port ' + port)
})

// 404 Error
app.use((req, res, next) => {
```

```
  next(createError(404));
});

app.use(function (err, req, res, next) {
  console.error(err.message);
  if (!err.statusCode) err.statusCode = 500;
  res.status(err.statusCode).send(err.message);
});
```

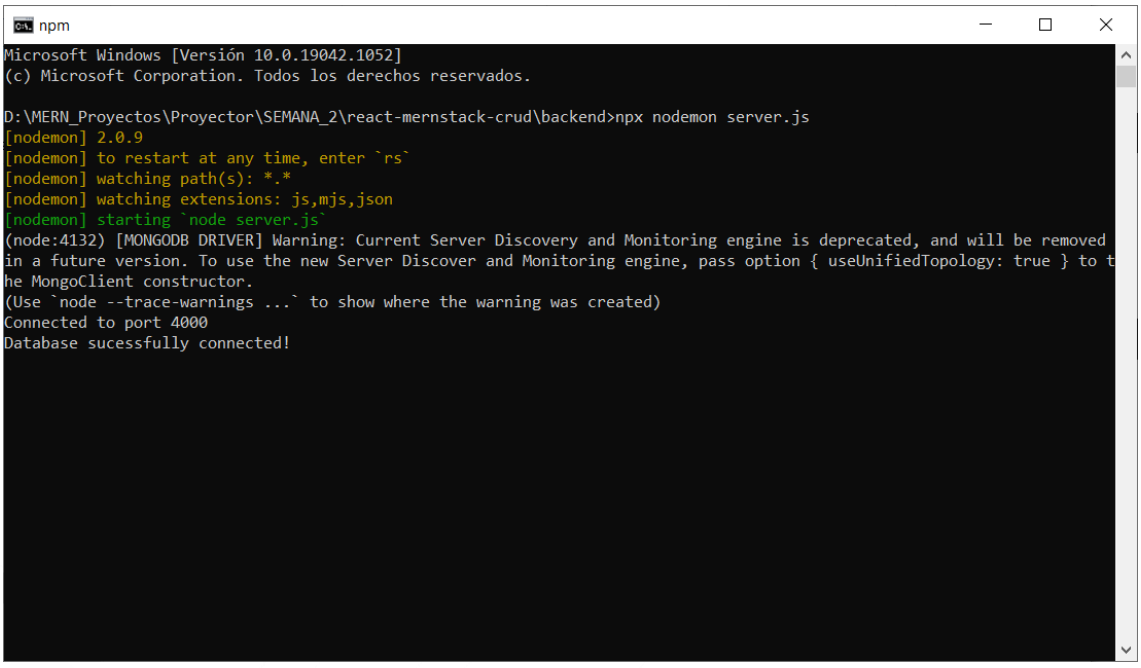
Ahora, hemos creado el backend para nuestra aplicación MERN CRUD. Abra un CMD en la carpeta backend y ejecute el comando para iniciar MongoDB, le permitirá guardar los datos del estudiante en la base de datos:

```
mongod
```

Seguidamente, abra otra terminal y ejecute el siguiente comando para iniciar el servidor Nodemon permaneciendo en la carpeta de backend.

```
npx nodemon server.js
```

Si ve el siguiente resultado en la pantalla de cmd significa que el node server está funcionando correctamente.



A continuación, se muestran las rutas de las API creadas con Express.js, MongoDB y Node.js.

REST API	URL
GET	http://localhost:4000/students
POST	/students/create-student
GET	/students/edit-student/id
PUT	/students/update-student/id
DELETE	/students/delete-student/id

"Prueba estas API en la herramienta de desarrollo de API Postmen."

Uso de Axios con React para realizar una solicitud HTTP

En este paso, aprenderemos a usar la biblioteca Axios en la aplicación React CRUD para manejar la solicitud HTTP. Axios es un cliente HTTP basado en promesas para el navegador y node.js. Ofrece las siguientes características.

- Hacer XMLHttpRequests desde el navegador
- Manejar solicitudes http de node.js
- Admite la API Promise
- Interceptar solicitud y respuesta
- Transformar los datos de solicitud y respuesta
- Cancelar solicitudes
- Autorregulación para datos JSON
- Protección del lado del cliente contra XSRF

Ejecute el comando en la terminal para instalar axios en la carpeta react-mernstack-crud

```
npm install axios
```

A continuación, enviaremos los datos del alumno al servidor MongoDB como un objeto utilizando el método HTTP post de Axios. Para ello, abrimos el archivo create-student.component.js y copiamos el siguiente código:

```
import React, { Component } from "react";
import Form from 'react-bootstrap/Form'
import Button from 'react-bootstrap/Button';
import axios from 'axios';

export default class CreateStudent extends Component {

  constructor(props) {
    super(props)
```

```
    // Setting up functions
    this.onChangeStudentName = this.onChangeStudentName.bind(this)
;
    this.onChangeStudentEmail = this.onChangeStudentEmail.bind(thi
s);
    this.onChangeStudentRollno = this.onChangeStudentRollno.bind(t
his);
    this.onSubmit = this.onSubmit.bind(this);

    // Setting up state
    this.state = {
      name: '',
      email: '',
      rollno: ''
    }
  }

  onChangeStudentName(e) {
    this.setState({ name: e.target.value })
  }

  onChangeStudentEmail(e) {
    this.setState({ email: e.target.value })
  }

  onChangeStudentRollno(e) {
    this.setState({ rollno: e.target.value })
  }

  onSubmit(e) {
    e.preventDefault()

    const studentObject = {
      name: this.state.name,
      email: this.state.email,
      rollno: this.state.rollno
    };
    axios.post('http://localhost:4000/students/create-
student', studentObject)
      .then(res => console.log(res.data));

   this.setState({
     name: '',
     email: '',
     rollno: ''
   });
```

```
  }

  render() {
    return (<div className="form-wrapper">
      <Form onSubmit={this.onSubmit}>
        <Form.Group controlId="Name">
          <Form.Label>Name</Form.Label>
          <Form.Control type="text" value={this.state.name} onChan
ge={this.onChangeStudentName} />
        </Form.Group>

        <Form.Group controlId="Email">
          <Form.Label>Email</Form.Label>
          <Form.Control type="email" value={this.state.email} onCh
ange={this.onChangeStudentEmail} />
        </Form.Group>

        <Form.Group controlId="Name">
          <Form.Label>Roll No</Form.Label>
          <Form.Control type="text" value={this.state.rollno} onCh
ange={this.onChangeStudentRollno} />
        </Form.Group>

        <Button variant="danger" size="lg" block="block" type="sub
mit">
          Create Student
        </Button>
      </Form>
    </div>);
  }
}
```

Luego ingrese el nombre del estudiante, el correo electrónico y rollno y haga clic en el botón "Crear estudiante" y sus datos se guardarán en la base de datos MongoDB NoSQL.

Mostrar lista de datos con React Axios

En este paso, mostraremos la lista de datos del estudiante usando React Axios y react bootstrap. Agregue el código que se proporciona a continuación dentro de `src/components/student-list.component.js`.

```
import React, { Component } from "react";
import axios from 'axios';
import Table from 'react-bootstrap/Table';
```

```
import StudentTableRow from './StudentTableRow';


export default class StudentList extends Component {

  constructor(props) {
    super(props)
    this.state = {
      students: []
    };
  }

  componentDidMount() {
    axios.get('http://localhost:4000/students/')
      .then(res => {
        this.setState({
          students: res.data
        });
      })
      .catch((error) => {
        console.log(error);
      })
  }

  DataTable() {
    return this.state.students.map((res, i) => {
      return <StudentTableRow obj={res} key={i} />;
    });
  }


  render() {
    return (<div className="table-wrapper">
      <Table striped bordered hover>
        <thead>
          <tr>
            <th>Name</th>
            <th>Email</th>
            <th>Roll No</th>
            <th>Action</th>
          </tr>
        </thead>
        <tbody>
          {this.DataTable()}
        </tbody>
      </Table>
```

```
    </div>);
  }
}
```

En el código anterior, estamos realizando una solicitud HTTP GET utilizando React Axios y Node/Express JS REST API. Estamos usando la React-Bootstrap table para mostrar los datos de los estudiantes en la interfaz.

En el siguiente paso, crearemos el componente y lo llamaremos `StudentTableRow.js` y lo guardaremos en la carpeta de `components`. Ya hemos importado el componente en el archivo `student-list.component.js`. Luego agregue el código que se proporciona a continuación dentro del archivo `components/StudentTableRow.js`.

```
import React, { Component } from 'react';
import { Link } from 'react-router-dom';
import Button from 'react-bootstrap/Button';

export default class StudentTableRow extends Component {
    render() {
        return (
            <tr>
                <td>{this.props.obj.name}</td>
                <td>{this.props.obj.email}</td>
                <td>{this.props.obj.rollno}</td>
                <td>
                    <Link className="edit-link" to={"/edit-
student/" + this.props.obj._id}>
                        Edit
                    </Link>
                    <Button size="sm" variant="danger">Delete</But
ton>
                </td>
            </tr>
        );
    }
}
```

Editar, actualizar y eliminar datos en React

En este paso, crearemos la funcionalidad de edición y actualización para que el usuario administre los datos de los estudiantes en React. Usamos la biblioteca Axios y realizamos la solicitud PUT para actualizar los datos en la base de datos MongoDB usando la API REST construida con Node y Express JS. Incluya el código que se proporciona a continuación dentro del archivo `edit-student.component.js`.

```
import React, { Component } from "react";
import Form from 'react-bootstrap/Form'
import Button from 'react-bootstrap/Button';
import axios from 'axios';

export default class EditStudent extends Component {

  constructor(props) {
    super(props)

    this.onChangeStudentName = this.onChangeStudentName.bind(this)
;
    this.onChangeStudentEmail = this.onChangeStudentEmail.bind(thi
s);
    this.onChangeStudentRollno = this.onChangeStudentRollno.bind(t
his);
    this.onSubmit = this.onSubmit.bind(this);

    // State
    this.state = {
      name: '',
      email: '',
      rollno: ''
    }
  }

  componentDidMount() {
    axios.get('http://localhost:4000/students/edit-
student/' + this.props.match.params.id)
      .then(res => {
        this.setState({
          name: res.data.name,
          email: res.data.email,
          rollno: res.data.rollno
        });
      })
      .catch((error) => {
        console.log(error);
      })
  }

  onChangeStudentName(e) {
    this.setState({ name: e.target.value })
  }

  onChangeStudentEmail(e) {
```

```
    this.setState({ email: e.target.value })
  }

  onChangeStudentRollno(e) {
    this.setState({ rollno: e.target.value })
  }

  onSubmit(e) {
    e.preventDefault()

    const studentObject = {
      name: this.state.name,
      email: this.state.email,
      rollno: this.state.rollno
    };

    axios.put('http://localhost:4000/students/update-
student/' + this.props.match.params.id, studentObject)
      .then((res) => {
        console.log(res.data)
        console.log('Student successfully updated')
      }).catch((error) => {
        console.log(error)
      })

    // Redirect to Student List
    this.props.history.push('/student-list')
  }


  render() {
    return (<div className="form-wrapper">
      <Form onSubmit={this.onSubmit}>
        <Form.Group controlId="Name">
          <Form.Label>Name</Form.Label>
          <Form.Control type="text" value={this.state.name} onChan
ge={this.onChangeStudentName} />
        </Form.Group>

        <Form.Group controlId="Email">
          <Form.Label>Email</Form.Label>
          <Form.Control type="email" value={this.state.email} onCh
ange={this.onChangeStudentEmail} />
        </Form.Group>

        <Form.Group controlId="Name">
```

```
          <Form.Label>Roll No</Form.Label>
          <Form.Control type="text" value={this.state.rollno} onCh
ange={this.onChangeStudentRollno} />
        </Form.Group>

        <Button variant="danger" size="lg" block="block" type="sub
mit">
          Update Student
        </Button>
      </Form>
    </div>);
  }
}
```

Realizar una solicitud de eliminación de Axios para eliminar datos

Por último, crearemos la función de eliminación en nuestra aplicación de demostración React CRUD. Vaya al archivo `components/StudentTableRow.js` y llame a la API Node/Express para eliminar los datos de los estudiantes de la base de datos MongoDB.

```
import React, { Component } from 'react';
import { Link } from 'react-router-dom';
import axios from 'axios';
import Button from 'react-bootstrap/Button';

export default class StudentTableRow extends Component {

    constructor(props) {
        super(props);
        this.deleteStudent = this.deleteStudent.bind(this);
    }

    deleteStudent() {
        axios.delete('http://localhost:4000/students/delete-
student/' + this.props.obj._id)
            .then((res) => {
                console.log('Student successfully deleted!')
            }).catch((error) => {
                console.log(error)
            })
    }

    render() {
        return (
            <tr>
```

```
                <td>{this.props.obj.name}</td>
                <td>{this.props.obj.email}</td>
                <td>{this.props.obj.rollno}</td>
                <td>
                    <Link className="edit-link" to={"/edit-
student/" + this.props.obj._id}>
                        Edit
                    </Link>
                    <Button onClick={this.deleteStudent} size="sm"
 variant="danger">Delete</Button>
                </td>
            </tr>
        );
    }
}
```

Aplicación Style CRUD

En este paso, debe diseñar la aplicación crud agregando el CSS personalizado en el archivo `src/App.css`:

```
.wrapper {
  padding-top: 30px;
}

body h3 {
  margin-bottom: 25px;
}

.navbar-brand a {
  color: #ffffff;
}

.form-wrapper,
.table-wrapper {
  max-width: 500px;
  margin: 0 auto;
}

.table-wrapper {
  max-width: 700px;
}

.edit-link {
  padding: 7px 10px;
```

```
  font-size: 0.875rem;
  line-height: normal;
  border-radius: 0.2rem;
  color: #fff;
  background-color: #28a745;
  border-color: #28a745;
  margin-right: 10px;
  position: relative;
  top: 1px;
}

.edit-link:hover {
  text-decoration: none;
  color: #ffffff;
}
```

Finalmente, editar el archivo src/App.test.js con el siguiente código:

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

it('renders without crashing', () => {
  const div = document.createElement('div');
  ReactDOM.render(<App />, div);
  ReactDOM.unmountComponentAtNode(div);
});
```