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
- Crear componentes en React
- Construir y trabajar con routers React
- Trabajar con React-Bootstrap
- Introducción a los formularios React
- Crear y consumir API REST con Express.js en la aplicación React
- Crear, Leer, Actualizar y Eliminar en React
- Configurar un servidor Node y Express
- Configurar MongoDB en el proyecto MERN
- Realizar solicitudes (promesas) HTTP con la biblioteca React Axios.

Prerequisitos

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 referencia.

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

Para construir la aplicación web MERN Stack, debe tener:

- Node.js: https://nodejs.org/es/download/
  - Documentación: https://nodejs.org/en/docs/
- MongoDB: https://www.mongodb.com/try/download/community2
  - Soporte Variables de entorno: https://www.youtube.com/watch?v=LibtQECAR1U
  - Documentación: https://www.mongodb.com/docs/
- Snippets React JS: https://www.youtube.com/watch?v=v4yPFSDJyJ0

Valide la instalación de su entorno

Una vez que el Node.Js y las variables de entorno estén correctas, ejecute en cmd el siguiente comando para verificar la versión de Node.js:

```
node -v
```

Una vez que el MongoDB y las variables de entorno estén correctas, ejecute en cmd el siguiente comando para verificar la versión de Node.js:

```
mongod -version
```

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:

```
npm 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 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.min.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:

(se puede utilizar el **Snippet = "rcc"** y cambiar el nombre de la clase)

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

export default class CreateStudent extends Component {
  render() {
    return <div>create-student.component</div>;
  }
}
```

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

(se puede utilizar el **Snippet = "rcc"** y cambiar el nombre de la clase)

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

export default class EditStudent extends Component {
  render() {
    return <div>edit-student.component</div>;
  }
}
```

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

(se puede utilizar el **Snippet = "rcc"** y cambiar el nombre de la clase)

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

export default class StudentList extends Component {
  render() {
    return <div>student-list.component</div>;
  }
}
```

Implementando React Router

En este paso, implementaremos `Router` en la aplicación React.js.

Ingrese el siguiente comando en cmd:

```
npm i react-router-dom@5.2.0 --save  (npm remove react-router-dom)
```

Archivo serviceWorker.js

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 users)
// will only see deployed updates on subsequent visits to a page, after all the
// existing tabs open on the page have been closed, since previously cached
// resources are updated in the background.

// To learn more about the benefits of this model and instructions on how to

const isLocalhost = Boolean(
  window.location.hostname === "localhost" ||
    // [::1] is the IPv6 localhost address.
    window.location.hostname === "[::1]" ||
    // 127.0.0.0/8 are 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 navigator) {
    // The URL constructor is available in all browsers that support SW.
    const publicUrl = new URL(process.env.PUBLIC_URL, window.location.href);
    if (publicUrl.origin !== window.location.origin) {
      // Our service worker won't work if PUBLIC_URL is on a different origin
      // from what our page is served on. This might happen if a CDN is used 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 ");
        });
      } 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 fetched,
              // but the previous service worker will still serve the older
              // content until all client tabs are closed.
              console.log(
                "New content is available and will be used when all "
              );

              // 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 page.
  fetch(swUrl, {
    headers: {
      "Service-Worker": "script",
    },
  })
    .then((response) => {
      // Ensure service worker exists, and that we really are getting a JS 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();
      })
      .catch((error) => {
        console.error(error.message);
      });
  }
}
```

Consulte el siguiente sitio web oficial para más información:

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

Archivo index.js

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")
);

serviceWorker.unregister();
```

Archivo App.js

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.min.css";
import "./App.css";

import { BrowserRouter as Router, Switch, Route, Link } from "react-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 (
    <div className="App">
      <Router>
        <header className="App-header">
          <Navbar bg="dark" variant="dark">
            <Container>
              <Navbar.Brand>
                <Link to={"/create-student"} className="nav-link">
                  App React MERN Stack
                </Link>
              </Navbar.Brand>

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

                <Nav>
                  <Link to={"/student-list"} className="nav-link">
                    Listar Estudiantes
                  </Link>
                </Nav>
              </Nav>
            </Container>
          </Navbar>
        </header>

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

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:

```
npm start
```

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

Crear formulario React con React Bootstrap

En este paso, crearemos el formulario utilizando el framework React Bootstrap para enviar los datos del estudiante en el componente `create-student.component.js`, de la forma:

```
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>Nombre</Form.Label>
            <Form.Control type="text" />
          </Form.Group>
          <Form.Group controlId="Email">
            <Form.Label>Correo Electrónico</Form.Label>
            <Form.Control type="email" />
          </Form.Group>
          <Form.Group controlId="Name">
            <Form.Label>Código</Form.Label>
            <Form.Control type="text" />
          </Form.Group>
          <Button variant="danger" size="lg" block="block" type="submit">
            Crear Estudiantes
          </Button>
        </Form>
      </div>
    );
  }
}
```

Recuerde ir a la página de React Bootstrap para incorporar los componentes de su formulario. Para mas información visite la siguiente página:

https://react-bootstrap.github.io/forms/overview/

Aplicación de estilos CRUD

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

```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;
}
```

Enviar datos del formulario en React

Primero, instalaremos `axios` (librería), que es un cliente HTTP basado en promesas, en consecuencia, diríjase a un `cmd` y en la carpeta raíz del proyecto ejecute el siguiente comando de consola:

```
npm install axios
```

Seguidamente, aprenderemos a enviar los datos del formulario 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.

1. Comenzaremos creando el constructor dentro de la clase (componente) CreateStudent.

(se puede utilizar el **Snippet = "con"** constructor predeterminado de la clase con props)

```
constructor(props) {
    super(props);
  }
```

2. Luego, dentro del constructor establecemos el estado inicial del componente CreateStudent configurando this.stateObject.

(se puede utilizar el **Snippet = "bnd"** método bind para cada atributo de la clase)

```
    this.onChangeStudentName = this.onChangeStudentName.bind(this);
    this.onChangeStudentEmail = this.onChangeStudentEmail.bind(this);
    this.onChangeStudentRollno = this.onChangeStudentRollno.bind(this);
    this.onSubmit = this.onSubmit.bind(this);
```

3. Seguidamente, dentro del constructor creamos los objetos de estados vacíos para inicializar cada atributo, de la forma:

(se puede utilizar el **Snippet = "est"** crear método de estados vacios)

```
    this.state = {
      name: "",
      email: "",
      rollno: "",
    };
```

4. Ahora, una vez finalizado el constructor, creamos los listener para definir que sucede cuando cambia el estado de un atributo, de la forma

(se puede utilizar el **Snippet = "on"** por cada atributo)

```
onChangeStudentName(e) {

}
onChangeStudentEmail(e) {

}
onChangeStudentRollno(e) {

}
```

5. Ahora, dentro de cada listener creamos el setter de cada atributo, de la forma:

(se puede utilizar el **Snippet = "sst"** establece el setter predeterminado para cada atributo)

```
onChangeStudentName(e) {
  this.setState({ name: e.target.value });
}
onChangeStudentEmail(e) {
  this.setState({ email: e.target.value });
}
onChangeStudentRollno(e) {
  this.setState({ rollno: e.target.value });
}
```

6. Luego, creamos el preventDefault() dentro del evento de envío onSubmit(e)

(se puede utilizar el **Snippet = "pd"**)

```
onSubmit(e) {
  e.preventDefault();

}
```

7. Ahora, dentro del preventDefault() creamos el objeto "student"

(se puede utilizar el **Snippet = "state"** para generar el estado del atributo)

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

8. Seguidamente, dentro del evento de envío (onSubmit) creamos la promesa, de la forma:

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

Recuerda importar `axios` al componente, de la forma:

```
import axios from "axios";
```

Finalmente, insertamos los listener dentro del formulario inicial, de la forma:

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

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

          <Form.Group controlId="Name">
            <Form.Label>Código</Form.Label>
            <Form.Control
              type="text"
              value={this.state.rollno}
              onChange={this.onChangeStudentRollno}
            />
          </Form.Group>

          <Button
            variant="danger"
            size="lg"
            block="block"
            type="submit"
            className="mt-4"
          >
            Crear Estudiantes
          </Button>
        </Form>
      </div>
    );
  }
}
```

El código final de “`create-student.component.js`” se muestra de la siguiente forma:

```
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);

    this.onChangeStudentName = this.onChangeStudentName.bind(this);
    this.onChangeStudentEmail = this.onChangeStudentEmail.bind(this);
    this.onChangeStudentRollno = this.onChangeStudentRollno.bind(this);
    this.onSubmit = this.onSubmit.bind(this);

    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>Nombre</Form.Label>
            <Form.Control
              type="text"
              value={this.state.name}
              onChange={this.onChangeStudentName}
            />
          </Form.Group>

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

          <Form.Group controlId="Name">
            <Form.Label>Código</Form.Label>
            <Form.Control
              type="text"
              value={this.state.rollno}
              onChange={this.onChangeStudentRollno}
            />
          </Form.Group>

          <Button
            variant="danger"
            size="lg"
            block="block"
            type="submit"
            className="mt-4"
          >
            Crear Estudiantes
          </Button>
        </Form>
      </div>
    );
  }
}
```

Construir el Backend Node JS para nuestro CRUD MERN Stack

Crearemos una carpeta dentro de nuestra aplicación para administrar los servicios "backend" como: base de datos, modelos, esquema, rutas y API. Para ello, creemos la siguiente carpeta:

`react-mernstack-crud/backend`.

Luego, necesitamos crear un archivo `package.json` en la carpeta: `react-mernstack-crud/backend` para administrar el backend de nuestra SPA. Para ello, ejecutamos el siguiente comando desde la consola y dentro de la carpeta "backend", de la forma:

```
npm init
```

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

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

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

```
npm i nodemon --save-dev
```

Troubleshooting:

Variables de entorno MongoDB:
https://javadesde0.com/tag/anadir-variables-entorno-mongodb/

No se reconoce comando nodemon:
https://www.wake-up-neo.net/es/javascript/el-comando-nodemon-no-se-reconoce-en-la-terminal-del-servidor-del-nodo-js/827560902/amp/

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

Asegúrese de que la propiedad "`main`" en este archivo tenga el nombre "`index.js`"

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.20.1",
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "mongoose": "^6.6.5"
  },
  "devDependencies": {
    "nodemon": "^2.0.20"
  }
}
```

Persistencia

Configuración de la base de datos MongoDB Atlas

A continuación, configuraremos una base de datos MongoDB Atlas (en la nube) para la aplicación React MERN. Para ello, visitemos la siguiente página y sigamos los pasos:

https://www.mongodb.com/es/cloud/atlas/efficiency

Para continuar con el registro, te recomiendo el siguiente video:

https://www.youtube.com/watch?v=6xxVCk4d4E0

Una vez ya estemos registrados, 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 = {
  mongoURI:
    "mongodb+srv://lmolero:12697883@cluster0.bpms5.mongodb.net/reactdb?retryWrit
es=true&w=majority",
  secretOrKey: "secret",
};
```

Hemos declarado la base de datos mongoDB llamada "`reactdb`".

Importante:

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 mongoose

Luego, crearemos 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.

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.

Ahora, crea 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();

// Modelo Estudiante
let studentSchema = require("../models/Student");

// CREAR Estudiante
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);
    }
  });
});

// LEER Estudiante
router.route("/").get((req, res, next) => {
  studentSchema.find((error, data) => {
    if (error) {
      return next(error);
    } else {
      res.json(data);
    }
  });
});

// Obtener un Estudiante
router.route("/edit-student/:id").get((req, res, next) => {
  studentSchema.findById(req.params.id, (error, data) => {
    if (error) {
      return next(error);
    } else {
      res.json(data);
    }
  });
});

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

// BORRAR Estudiante
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 en el backend

Casi hemos creado todo para configurar el backend de Node y Expresss.js para la aplicación React CRUD.

Ahora crearemos el archivo index.js en la siguiente dirección: `react-mernstack-crud/backend/`

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

```
let express = require("express");
let mongoose = require("mongoose");
let cors = require("cors");
let bodyParser = require("body-parser");

// Ruta de Express
const studentRoute = require("../backend/routes/student.route");

// DB Config
const db = require("../backend/database/db").mongoURI;
// Connect to MongoDB from mLab
mongoose
  .connect(db, { useNewUrlParser: true })
  .then(() => console.log("MongoDB successfully connected"))
  .catch((err) => console.log(err));

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);
});

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

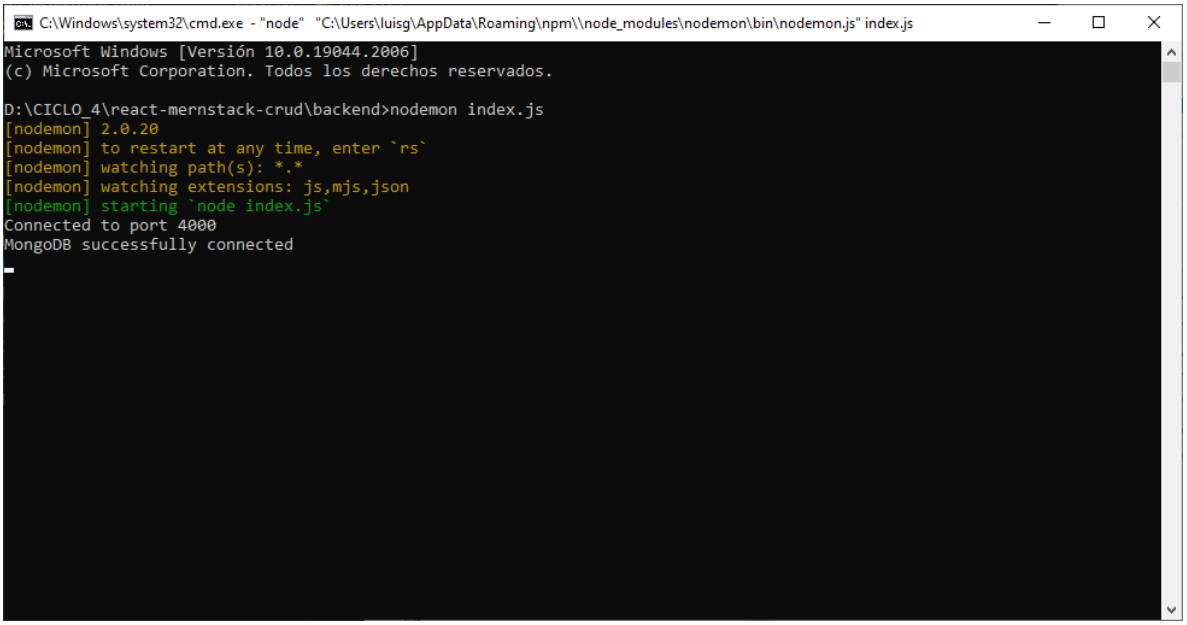
Ahora, hemos creado el backend para nuestra aplicación MERN CRUD. Abra un CMD en la carpeta `backend` y ejecute el comando para iniciar MongoDB, le permitirá guardar los datos del estudiante en la base de datos:

```
mongod
```

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

```
nodemon index.js
```

Si ve el siguiente resultado en la pantalla de cmd significa que `index.js` está funcionando correctamente.



Rutas de las API creadas con Express.js, MongoDB y Node.js.

API REST	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 Postman. Descarga Postman aquí:

https://www.postman.com/downloads/

Mostrar lista de datos

Ahora, crearemos un componente que llamaremos `StudentTableRow.js` y lo guardaremos en la carpeta de `components`. Este componente, permitirá renderizar una grilla con todos los elementos de la bd, asimismo, tendrá axios para manejo de promesas para peticiones de “edición” y “borrado” de registros que se podrá habilitar desde sus botones.

En consecuencia, pegue el siguiente código en el archivo `StudentTableRow.js`, de la forma:

```
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"
            path={"product/:id"}
            to={"/edit-student/" + this.props.obj._id}
          >
            Editar
          </Link>
          <Button onClick={this.deleteStudent} size="sm" variant="danger">
            Borrar
          </Button>
        </td>
      </tr>
    );
  }
}
```

Seguidamente, En este paso, mostraremos la lista de datos del estudiante usando React Bootstrap desde el componente `student-list.component.js`. 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>Nombre</th>
              <th>Correo Electrónico</th>
              <th>Código</th>
              <th>Acción</th>
            </tr>
          </thead>
          <tbody>{this.DataTable()}</tbody>
        </Table>
      </div>
    );
  }
}
```

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(this);
    this.onChangeStudentRollno = this.onChangeStudentRollno.bind(this);
    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>Nombre</Form.Label>
            <Form.Control
```

```
            type="text"
            value={this.state.name}
            onChange={this.onChangeStudentName}
          />
        </Form.Group>

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

        <Form.Group controlId="Name">
          <Form.Label>Código</Form.Label>
          <Form.Control
            type="text"
            value={this.state.rollno}
            onChange={this.onChangeStudentRollno}
          />
        </Form.Group>

        <Button variant="danger" size="lg" block="block" type="submit">
          Actualizar Estudiante
        </Button>
      </Form>
    </div>
  );
 }
}
```

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

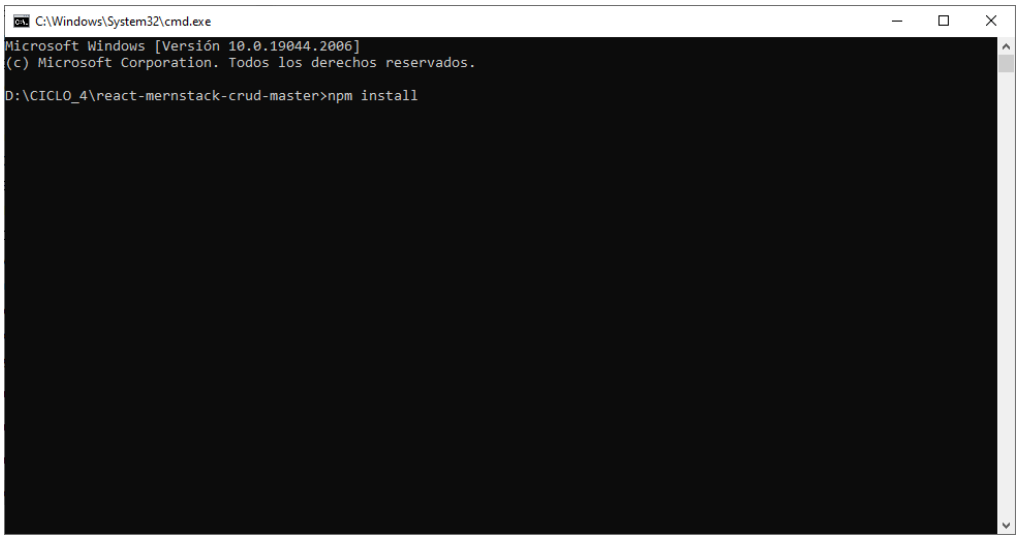
```
import { render, screen } from "@testing-library/react";
import App from "./App";

test("renders learn react link", () => {
  render(<App />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

Instalar y ejecutar la SPA

Si tienes el código de la App y quieres ejecutarlo debes hacer los siguientes pasos en una consola cmd:

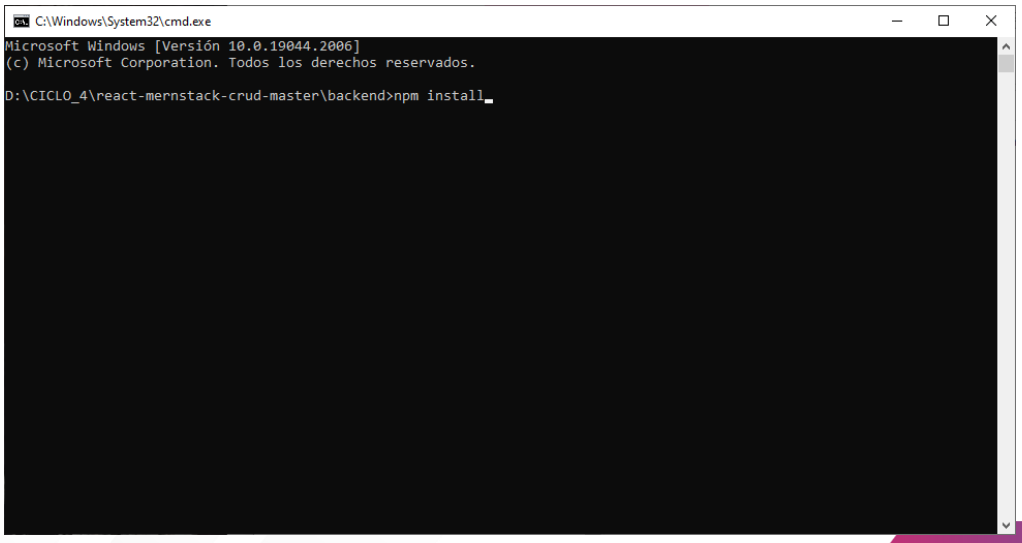
npm install dentro de la carpeta raíz:

```
npm install
```



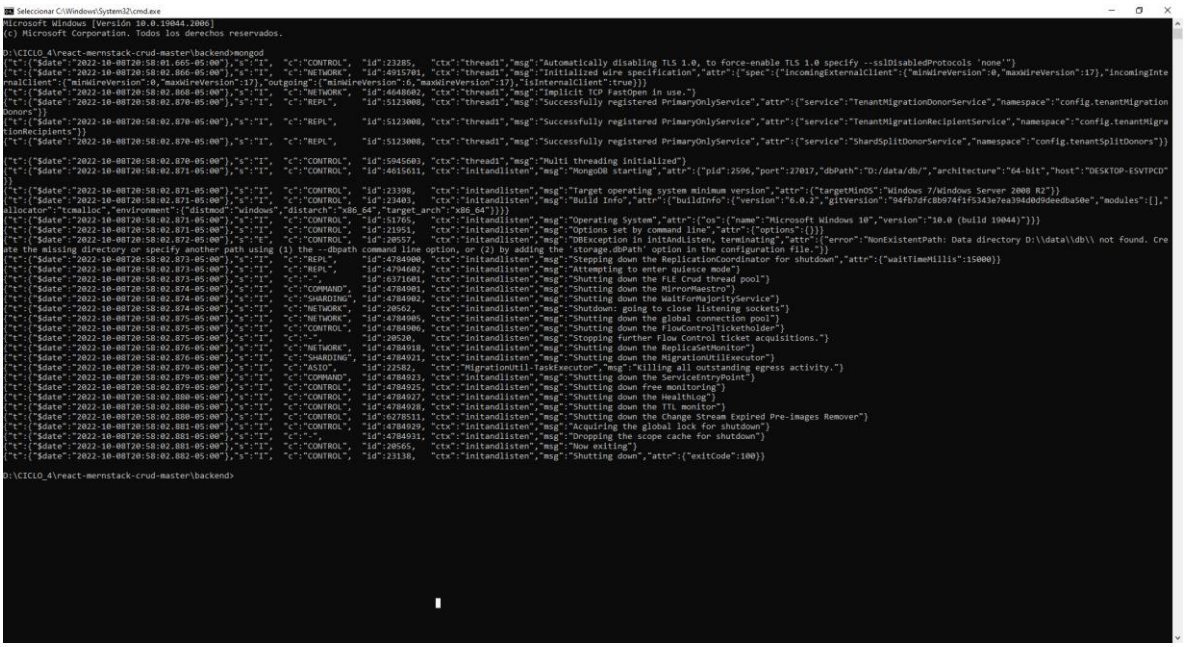
Ahora, npm install dentro de la carpeta “backend”:

```
npm install
```



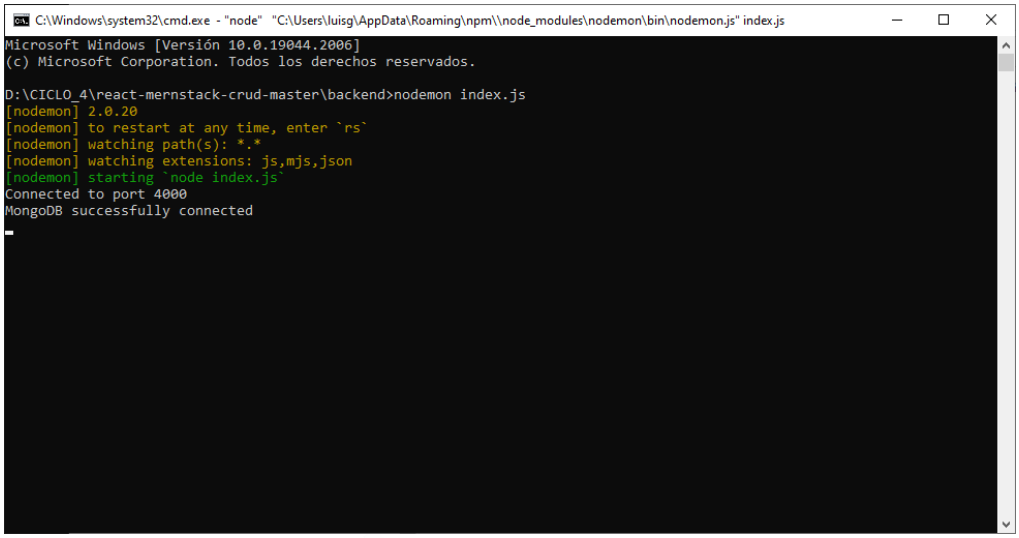
Seguidamente, entra en la carpeta “backend” y ejecuta en 2 consolas diferentes los siguientes comandos:

```
mongod
```



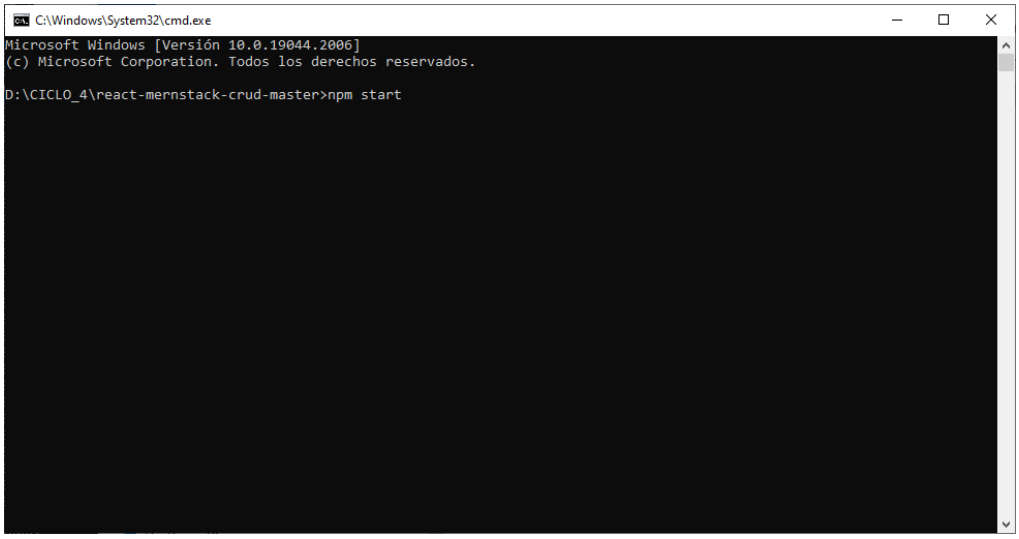
y, seguidamente:

```
nodemon index.js
```



Finalmente, ejecuta en la carpeta “raíz” el comando npm start, de la forma

```
npm start
```



¡¡Éxitos!!