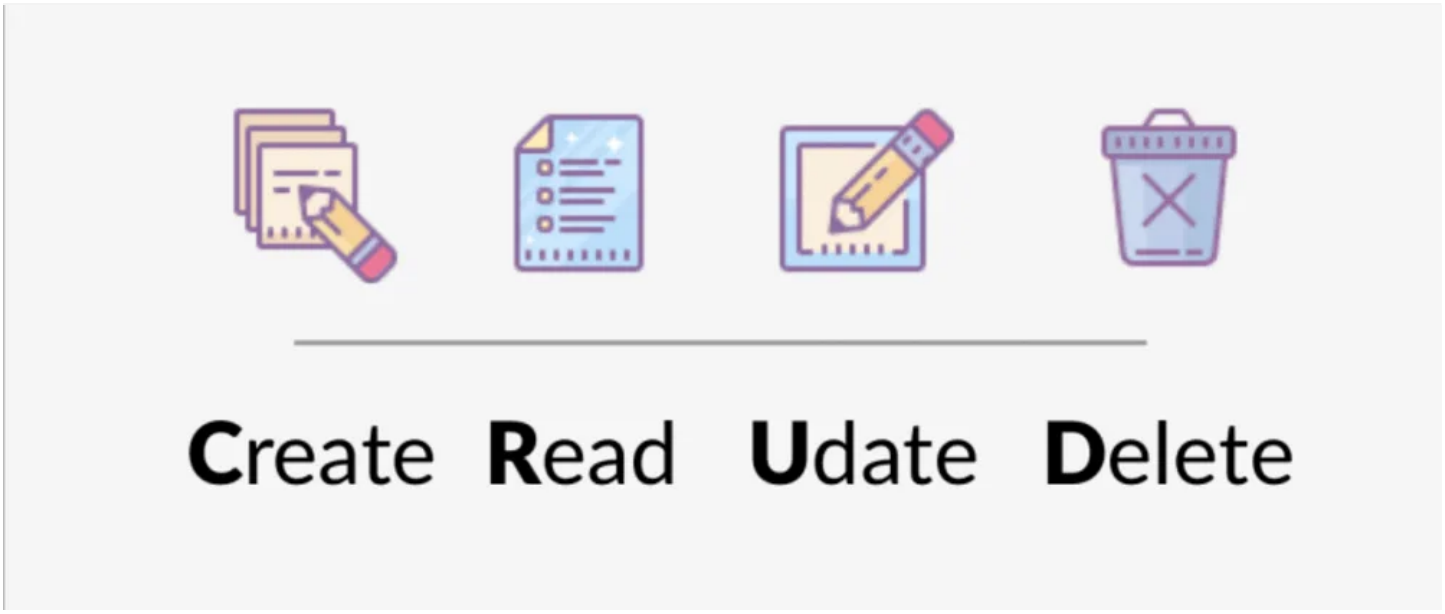


Ruhul Amin
Posted on 30 de dez. de 2021 • Updated on 12 de mai.

What is CRUD Operation? How to Build CRUD Operations using React, Node, Express, and MongoDB?

#crud #react #nore #mongodb

CRUD:

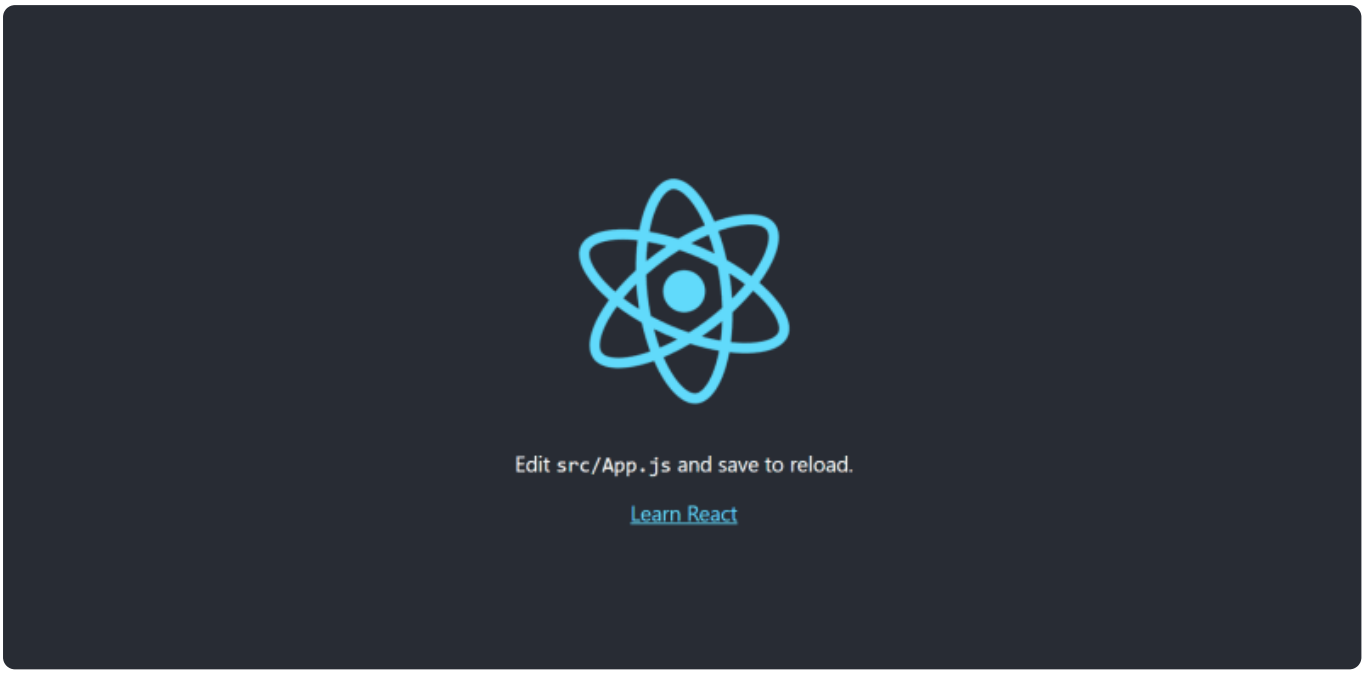
CRUD stands for Create, Read, Update, and Delete. CRUD typically refers to operations performed in a database. When an application is connected to a database, it adds data to the database, reads the data, updates any data can delete the data is called CRUD operation.

Create — To insert any record into the database.
Read — To retrieve records from the database.
Update — To update a record in the database.
Delete — To delete a record in the database

How to Build your CRUD Application:

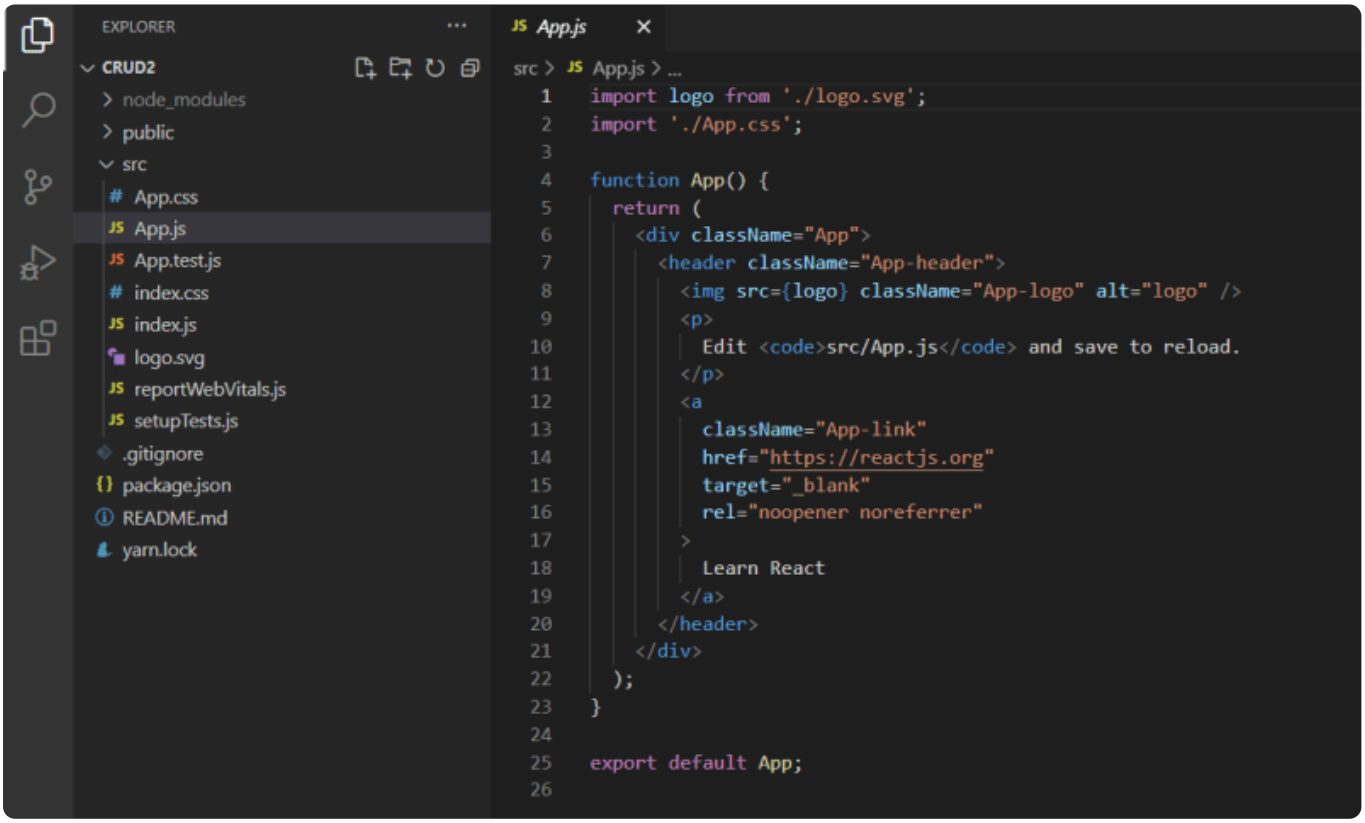
To create a CRUD operation first of all we need to create a react application. To create your React application, type `npx create-react-app <Your app name>` in your terminal.

You'll see that the packages are being installed. After creating this react project go to the project folder and open it, then open the terminal and command `npm start`. Now you will see the default React template, like this:



That means you successfully created a react application.

Now we will go to your code editor and open the app.js file You will see a default boilerplate like this



Now we need to create another project for the server. Before installing the server environment you must install node.js software on your computer.

Now you can create a directory on your computer for the server, and open the directory in your terminal. Now you can create a server following steps.

- npm init -y
- Npm install express cors mongodb dotenv nodemon

Insert the 2 following lines in your scripts property in your package.json file

```
"scripts": {
    "start": "node index.js",
    "start-dev": "nodemon index.js"
```

Now open the project create a file named index.js and insert other necessary things.

```
const express = require("express");
const { MongoClient } = require("mongodb");
require("dotenv").config();
const cors = require("cors");
const ObjectId = require("mongodb").ObjectId;
const app = express();
const port = process.env.PORT || 5000;
```

Must Use middleware in your `index.js` file

```
app.use(cors());
app.use(express.json());
```

And then create your MongoDB cluster and input your `.env` file and use it on your `index.js` file like this.

```
const uri = `mongodb+srv://${process.env.DB_USER}:${process.env.DB_PASS}@cluster0.qu
```

Now we create a function like the following code and create a database and connect with the database. write everything in the try block.

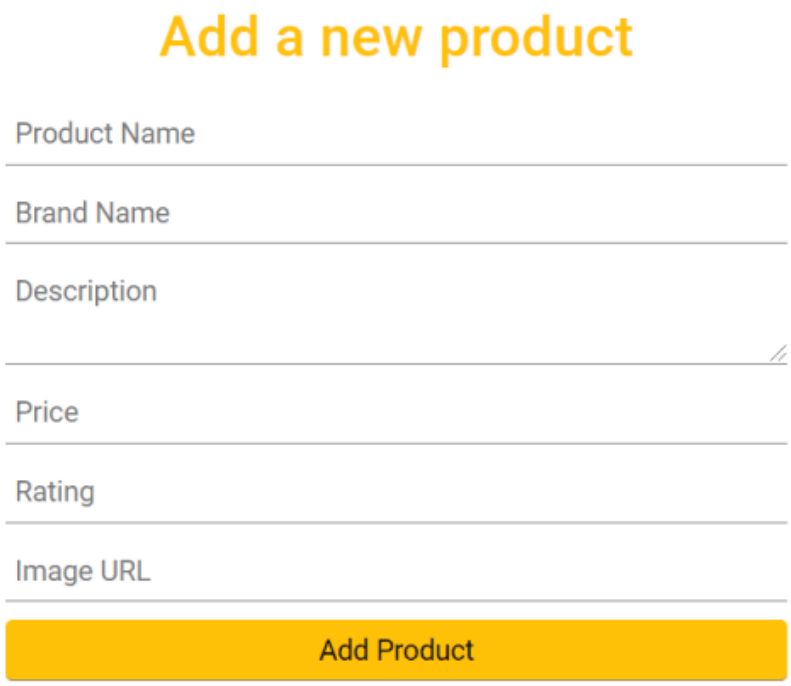
```
async function run() {
  try {
await client.connect();
    const database = client.db("modernFurniture");
    const productsCollection = database.collection("products");
    const ordersCollection = database.collection("orders");
    const usersCollection = database.collection("users");
    const reviewsCollection = database.collection("reviews");
```

```
  } finally {
    // await client.close();
  }
}
run().catch(console.dir);
```

Let's start CRUD Operations:

Create Operation:

Now we create an HTML input form for getting some data and send it to the server for the Create operation.



We use Axios and react hook form to send data to the server. For installing Axios write command on your terminal `npm i axios`, and for install react hook form `npm install react-hook-form`

```
const AddProduct = () => {
  const { register, handleSubmit, reset } = useForm();
  const onSubmit = (data) => {
    axios
   .post("http://localhost:5000/products", data)
      .then((res) => {
        "Do something"        }
```

```
    });
};
```

Now we create a post API on the node server to create data into the database.

```
app.post("/products", async (req, res) => {
  const product = req.body;
  const result = await productsCollection.insertOne(product);
  res.json(result);
 });
```

Read Operation:

Now we create a component on our client-side and send a request to the server for getting all data to show our browser. And when the server responds then stores the data in a state. And we can show data on the UI.

Create a get request to get data from the database:

```
const [products, setProducts] = useState([]);
useEffect(() => {
  fetch("http://localhost:5000/products")
    .then((res) => res.json())
    .then((data) => setProducts(data));
}, []);
```

Sending data to the browser from the server:

```
// GET API
 app.get("/products", async (req, res) => {
  const cursor = productsCollection.find({});
  const products = await cursor.toArray();
  res.json(products);
 });
```

Update Operation:

If further need to update the existing data then we use to update operation. First, we create an update request on our client-side with the following code.

```
const user = { email };
   fetch("http://localhost:5000/users/admin", {
     method: "PUT",
     headers: {
       "content-type": "application/json",
     },
     body: JSON.stringify(user),
   })
     .then((res) => res.json())
     .then((data) => {
       "do something";
     });
```

Then the response from the server:

```
  // make a admin user
   app.put("/users/admin", async (req, res) => {
     const user = req.body;
     const filter = { email: user.email };
     const updateDoc = { $set: { role: "Admin" } };
     const result = await usersCollection.updateOne(filter, updateDoc);
     res.json(result);
   });
```

We make the normal user to an admin user use that above code.

Delete Operation:

When we need to delete any data from the database then we use the delete operation. There we want to delete a customer order.

```
const url = `http://localhost:5000/orders/${id}`;
        fetch(url, {
          method: "DELETE",
        })
          .then((res) => res.json())
          .then((data) => {
            if (data.deletedCount > 0) {
```

```
                alert("Deleted!", "Your order has been deleted.", "success");
              }
            });
```

Server response:

```
// delete single order
  app.delete("/orders/:id", async (req, res) => {
    const id = req.params.id;
    const query = { _id: ObjectId(id) };
    const result = await ordersCollection.deleteOne(query);
    res.json(result);
  });
```

Now the server response and the order delete from ui and delete from the database also.

Top comments (0)

Ruhul Amin

Web Developer

LOCATION
Dhaka, Bangladesh

EDUCATION
Bangladesh University

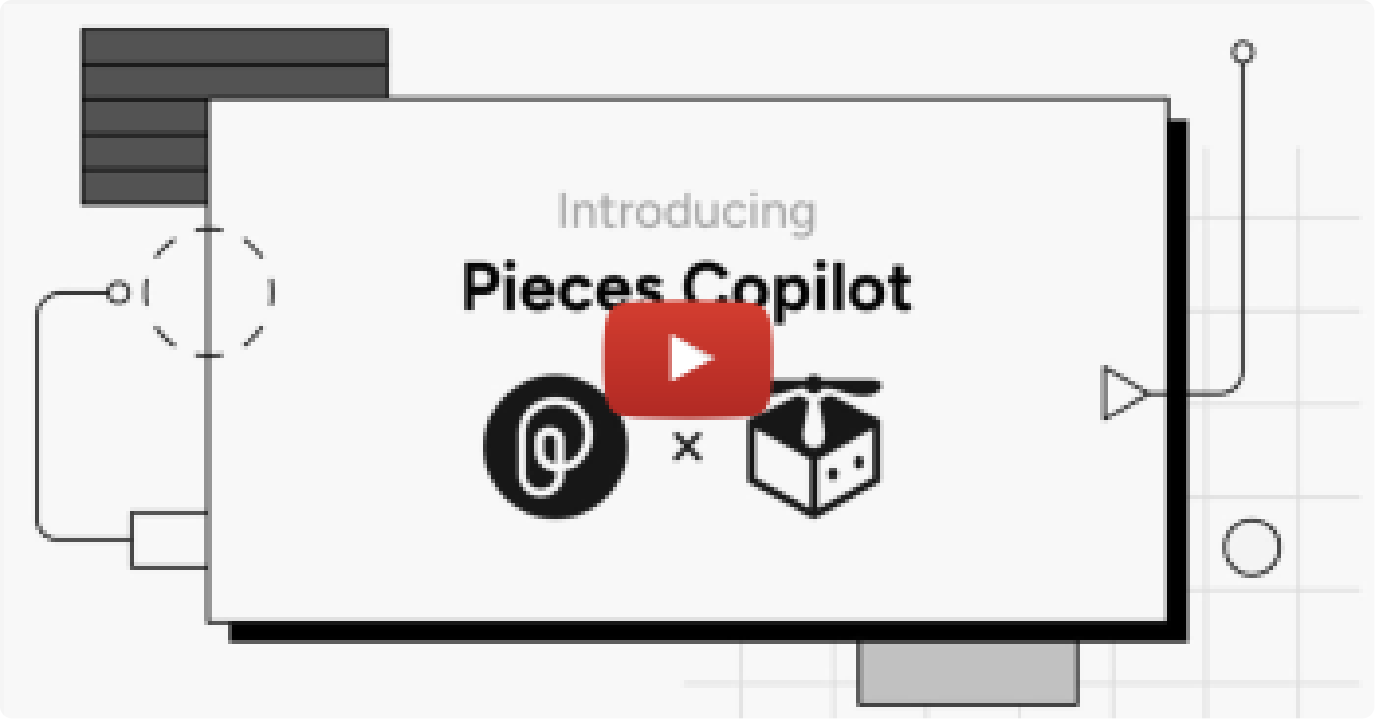
WORK
MERN Stack Web Developer

JOINED
29 de dez. de 2021

More from Ruhul Amin

What is Redux? Why we should use it?
#redux #react #javascript #webdev

What is context API? why we should use it? How to use it in react?
#react #context #webdev



Imagine a tool that integrates seamlessly across your entire development workflow - from browsing solutions online to writing code in your IDE, and even during team collaborations. Pieces is that tool (or tool-between-tools).

Read full post