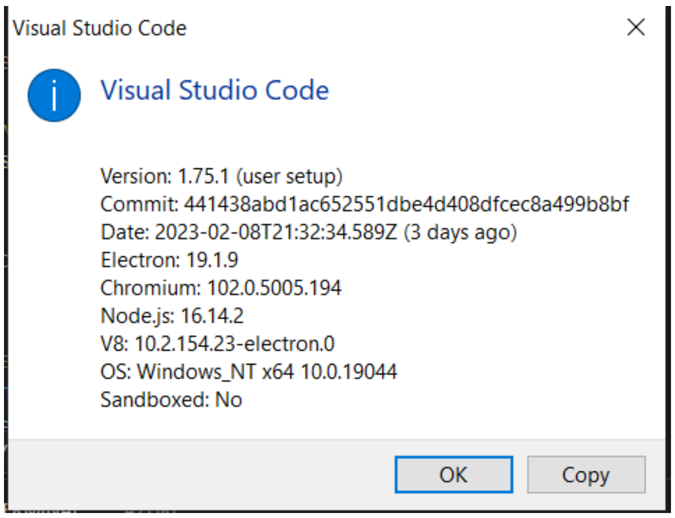
Step by Step Guide to Create MERN Application

Created by: Bahadur Singh, 12 Feb 2023

This document will explain you steps about creating a small MERN (MongoDB, Express, React and Node JS) technologies based application. You will learn the CRUD operations (Create, Read, Update and Delete).

What is needed?

Visual Studio Code with below components.
Node JS installed
MongoDB install on your local computer. You can also use cloud base ATLAS version.
I am using this on Windows 10.



```
Version: 1.75.1 (user setup)
Commit: 441438abd1ac652551dbe4d408dfcec8a499b8bf
Date: 2023-02-08T21:32:34.589Z (3 days ago)
Electron: 19.1.9
Chromium: 102.0.5005.194
Node.js: 16.14.2
V8: 10.2.154.23-electron.0
OS: Windows_NT x64 10.0.19044
Sandboxed: No
```

Command you needed to know on node JS Terminal.

>> npm install node

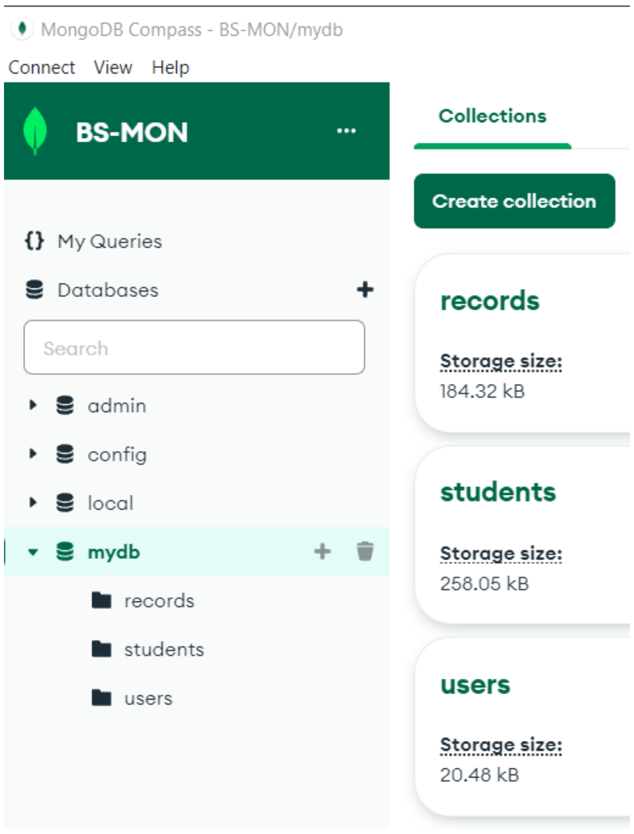
>> npm install express –save

>> npm install axios –save

>> npm install mongodb

>> npm install mongoose

MongoDB Setup



In this document, I will be using the collections under mydb (records or students)

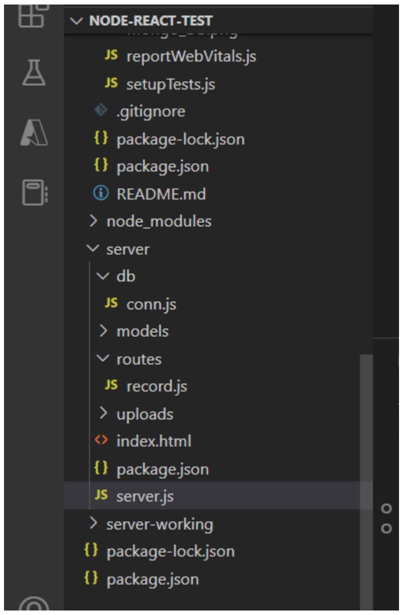
Records have name, position, level, and Image fields. Image data is saved as base64 string.

MongoDB is noSQL Database. More documentation and details you can see mongodb.com

Collection **records** is used with this schema:

```
{
        name : string
        position: string
        level: string
        imgDataStr : string (base64 string)
}
```

Server Side Application Structure:

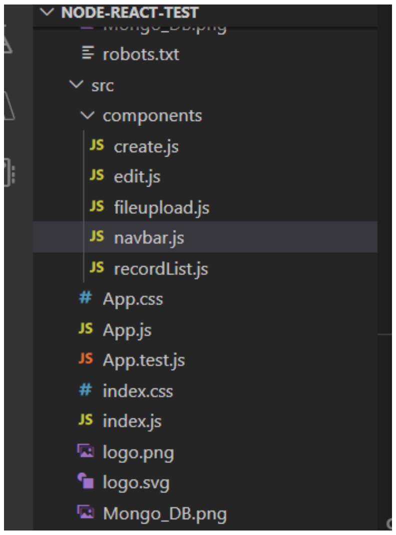


```
NODE-REACT-TEST
    reportWebVitals.js
    setupTests.js
  .gitignore
  package-lock.json
  package.json
  README.md
> node_modules
v server
  v db
    conn.js
  > models
  v routes
    record.js
  > uploads
  index.html
  package.json
  server.js
> server-working
package-lock.json
package.json
```

Server.js => This is master main driver file to launch a Server at Port 5000 or configured value.
db / conn.js => connection object to MongoDB database using Mongoose
routes / record.js => This file has server API for the client to connect to. Like CRUD operations. Read All records, Create Record, Update and Delete record.

Client Side Application (React JS)

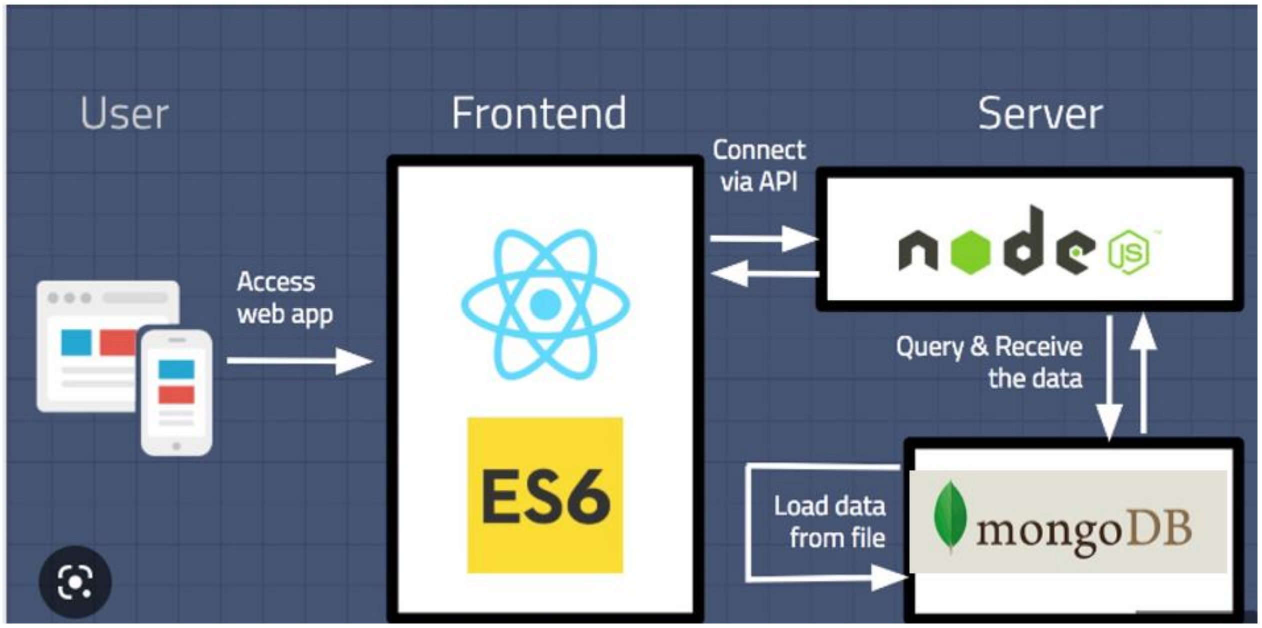
React JS application runs also in its own server localhost:3000 port when you install and initialize it. Its powerful front end application and is way good stronger than other front ends. Lost of good programming trick options, styling is automatically available, in just few minutes, you can build a good CRUD application.



```
NODE-REACT-TEST
  robots.txt
v src
  v components
    create.js
    edit.js
    fileupload.js
    navbar.js
    recordList.js
  App.css
  App.js
  App.test.js
  index.css
  index.js
  logo.png
  logo.svg
  Mongo_DB.png
```

App.js => This is main driver module of React JS front end. It launches an browser base HTML rendered application show the data records from Mongo DB.

Architecture Concept



Code Explanation & Details

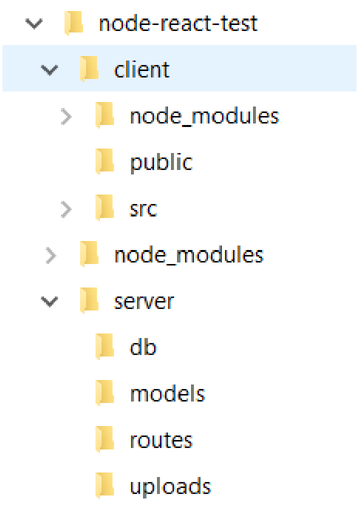
Lets understand first the server side. Goto Visual Studio Code, create a directory say node-react-test.

On this directory, create a server folder. For Server side code

On this directory create a client folder. For Client Side Front End Code, see in later section.

Change directory to server folder and run

>> npm init -y command, it will do needful setup



```
node-react-test
  client
    node_modules
    public
    src
  node_modules
  server
    db
    models
    routes
    uploads
```

Server Side Code and Implentation

Server.js:

```
const express = require("express");
const app = express();
const cors = require("cors");
var path = require ('path');
require("dotenv").config({ path: "./config.env" });
const port = process.env.PORT || 5000;
app.use(cors());
app.use(express.json());
app.use(require("./routes/record"));
// For file upload from browser or external client
app.use('/uploads/', express.static(path.join(__dirname, './', 'uploads')))
// get DB driver connection
const dbo = require("./db/conn");
app.listen(port, () => {
  //perform a database connection when server starts
    dbo.connectToServer( function(err) {
    if (err) console.error(err);
      dbo.println ('server,js', "Database connection is opened");
    });
    dbo.println('server.js', 'Server is running on port:'+ `${port}`);
});
```

Conn.js => This is an interface to database object. You need MongoDB installed as an application. And mongodb and mongoose modules install inside the node terminal console

```
const { MongoClient } = require("mongodb");
const url = "mongodb://localhost:27017/";

const client = new MongoClient(url, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

var _db;

module.exports = {
 connectToServer: function (callback) {
    console.log ("conn.js connecting to MongoDB.", url);
    client.connect(function (err, db) {
      // Verify we got a good "db" object
      //console.log(" connecting to MongoDB." +  db);
      if (db)
      {
        _db = db.db("mydb");
       console.log ('conn.js Successfully connected to MongoDB.');
      }
      return callback(err);
    });
  },

  getDb: function () {
    //console.log (" getDb called ");
    return _db;
  },

  println: function (cls, msg, msg1="")
  {
  var jsonDate = (new Date()).toJSON();
  console.log (jsonDate + " "+ cls + " :: " , msg, msg1 );
  },

};
```

Now we will define the routes. The server API will open these functions. We will define recordRoutes functions that will expose middle-ware server API for the React Client. Server application will be run on PORT 5000, it will connect to MongoDB for DB CRUD operations. Express will be running in middle-ware to provide an infrastructural application framework.

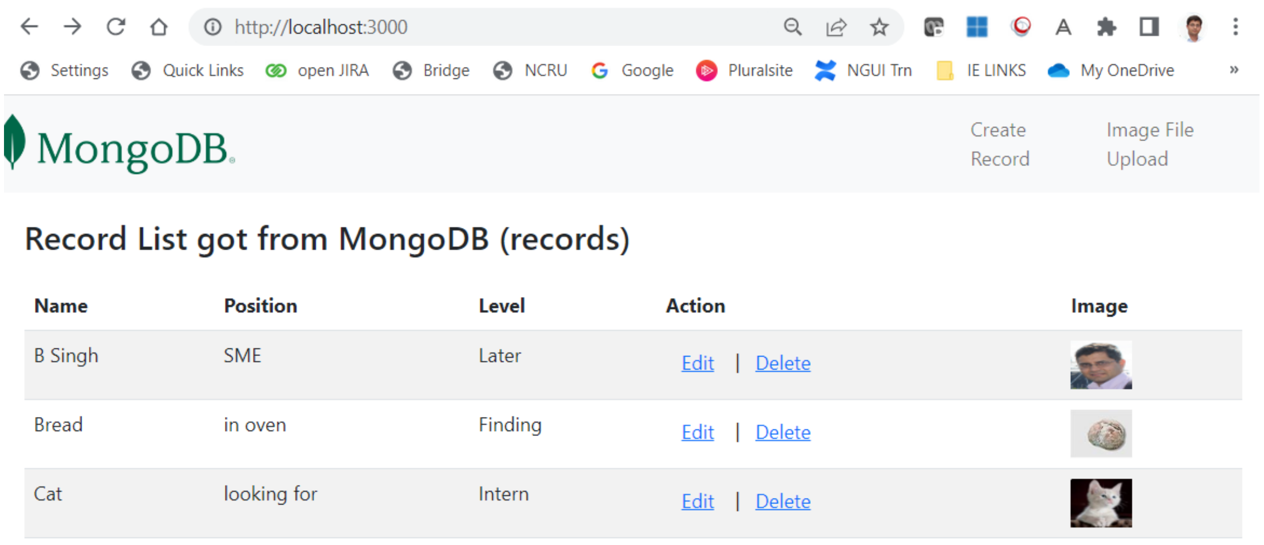
React Client < --- localhost:3000 --- > connects to Server API < --- localhost:5000 -- > connects Mongo DB

Routes we have defined for: /record, /record:id, record/add, /update:id, and /:id delete operation. Follow the code, its running example. There is also an independent upload file API as well. It copies the PNG file from client and put on the server. This uploaded file is attached as picture to the record.

```
const express = require("express");
const fs = require ("fs");
const formidable = require ("formidable"); // file upload component
const path = require("path");

var tDataBuf;
var imageName;

// recordRoutes is an instance of the express router.
// We use it to define our routes.
// The router will be added as a middleware and will take control of requests
starting with path /record.
const recordRoutes = express.Router();

// This will help us connect to the database
```

```
const dbo = require("../db/conn");

// This help convert the id from string to ObjectId for the _id.
const ObjectId = require("mongodb").ObjectId;

// This section will help you get a list of all the records.
recordRoutes.route("/record").get(function (req, res) {
  //console.log ("server API: read records called. ");
  let db_connect = dbo.getDb("mydb");
  db_connect
    .collection("records")
    .find({})
    .toArray(function (err, result) {
      if (err) throw err;
      res.json(result);
    });
});

// This section will help you get a single record by id
recordRoutes.route("/record/:id").get(function (req, res) {
  let db_connect = dbo.getDb();
  let myquery = { _id: ObjectId( req.params.id )};
  db_connect
      .collection("records")
      .findOne(myquery, function (err, result) {
        if (err) throw err;
        res.json(result);
      });
});

// This section will help you create a new record.
recordRoutes.route("/record/add").post(function (req, response) {
  console.log ("serverAPI add tDataBuf imageName " + imageName );
  if (tDataBuf) console.log (`${tDataBuf.length}`);

  let db_connect = dbo.getDb();
  let myobj = {
    name: req.body.name,
    position: req.body.position,
    level: req.body.level,
    imgDataStr : tDataBuf,
  };
  db_connect.collection("records").insertOne(myobj, function (err, res) {
    if (err) throw err;
    response.json(res);
  });
  tDataBuf = "";
});

// This section will help you update a record by id.
recordRoutes.route("/update/:id").post(function (req, response) {
  let db_connect = dbo.getDb();
  let myquery = { _id: ObjectId( req.params.id )};

  console.log ("serverAPI update tDataBuf imageName " + imageName );
  if (tDataBuf) console.log (`${tDataBuf.length}`);

  let newvalues = {
    $set: {
      name: req.body.name,
      position: req.body.position,
      level: req.body.level,
      imgDataStr: tDataBuf,
    },
  };

  db_connect
    .collection("records")
    .updateOne(myquery, newvalues, function (err, res) {
      if (err) throw err;
      console.log("server API: 1 document updated ", res.modifiedCount);
      response.json(res);
    });
    tDataBuf = "";
});

// This section will help you delete a record
recordRoutes.route("/:id").delete((req, response) => {
  let db_connect = dbo.getDb();
  let myquery = { _id: ObjectId( req.params.id )};
  db_connect.collection("records").deleteOne(myquery, function (err, obj) {
    if (err) throw err;
    console.log("1 document deleted ", obj.deletedCount, myquery);
    response.json(obj);
  });
});

// This is server API end point for getting the file from browser and put to
uploads folder
// app.use (./uploads) folder is needed otherwise server does not know where
to copy it.
// formidable is needed. readfile is incoming file's name
// router is needed to define to have this /upload defined, so that
http://localhost:5000/upload
// is open to React JS for puttign the file in it. It can be also tested from
POSTMAN Tool
 recordRoutes.route("/upload").post( async function (req, response) {
  //var inputData= req.body;
  console.log ("server API : uploadFile called.. ");

  const form = new formidable.IncomingForm ();
  form.parse (req, function (err, fields, files) {
     // got the file from brower form and now available in temp dir. will be
written to new dir
     //console.log (files);

     let oldPath = files.readfile.filepath; // this is native raw file
(readfile) is name of field passed from Form
     console.log (" uploaded temp Path " , oldPath);
     let newPath = path.join(__dirname, '../uploads') + '/' +
files.readfile.originalFilename;
     var rawData = fs.readFileSync(oldPath);
     tDataBuf =  Buffer.from(rawData).toString('base64');
     imageName = files.readfile.originalFilename;
     fs.writeFile (newPath, rawData, function (err) {
        if (err) console.log ("error writing file ", err.message);
        else console.log ("file written successfully ", fields);
     });

     console.log ("newpath" , newPath);
     return response.status (200).json ({state: 'uploaded ' + imageName});
    });

    console.log ("serverAPI uploadFile Over.. ");
});

module.exports = recordRoutes;
```

/uploads folder, you need to define for uploading the files. These files can be used as static link for reference on the web-page. My example shows an operation to write the PNG data to Database in base64 format and then read it back and render on the Browser.



Record List got from MongoDB (records)

Name	Position	Level	Action	Image
B Singh	SME	Later	Edit | Delete	
Bread	in oven	Finding	Edit | Delete	
Cat	looking for	Intern	Edit | Delete	

This is an example of this client. But we are not done yet. We just completed the server side task. Now we have to implement the client side work. The real React JS fun…

Client Side Code and Implementation.

You have seen above the directory structure about how to do the sample setup.

We need to start with following, change directory to client.

It will create an environment for react js.

>> npm react-create-app client

>> cd client

>> npm start

The program will create some files and folder and App.js and/or index.js.

src/App.js => It is my driver program to launch the front end. It has an App() function call that will invoke my defined Routes to Create / Update / Read All and Delete functions.

```
// Main Start Point of React JS program.
//
import React from "react";

// We use Route in order to define the different routes of our application
import { Route, Routes } from "react-router-dom";

// We import all the components we need in our app
import Navbar from "./components/navbar";
import RecordList from "./components/recordList";
import Edit from "./components/edit";
import Create from "./components/create";
import UploadFile from "./components/fileupload";

// Application start, its defining the navigation
const App = () => {
  return (
    <div>
      <Navbar />
      <div style={{ margin: 20 }}>
      <Routes>
        <Route exact path="/" element={<RecordList />} />
        <Route path="/edit/:id" element={<Edit />} />
        <Route path="/create" element={<Create />} />
        <Route path="/upload" element={<UploadFile />} />
      </Routes>
      </div>
    </div>
  );
};

export default App;
```

Now we will move to src/components. It has these files:

Create.js => For creating new record in database.

Edit.js => This is for the editing / update the record in database

Recordlist.js => this is for “/” root loading of the page to show all the records from database.

Delete function is directly invoked from the browser button function using the _id of the object to delete. Easy.. not much code is needed. See Server API .delete function how it is deleted.

Nav.js => This function is used for defining navigation UI on front-end. It has not much logic, just a call to create page, edit page and file upload page.

Fileupload.js => This is functionality for uploading file from browser to server. It is saved under /uploads folder.

At React JS front end, few modules needed. Like axios for file upload. Plain fetch command did not work for me. On the server side code, you need formidable module. Some people use mutler or others, I have not tried those.

Running Code sample:

Create.js:
```
import React, { useState } from "react";
import { useNavigate } from "react-router";
import axios from "axios";

// create an empty record
export default function Create() {
  const [form, setForm] = useState({
    name: "",
    position: "",
    level: "",
  });

  const [file, setFile] = useState (); // save file info.

  const navigate = useNavigate();

  // These methods will update the state properties.
  function updateForm(value) {
    return setForm((prev) => {
      return { ...prev, ...value };
    });
  }

  // This function will handle the submission.
  async function onSubmit(e) {
    e.preventDefault();

    // first upload the Image file, so that can be used for the record
creation.

    //handleFileUpload ();
      const url = 'http://localhost:5000/upload';
      const formData = new FormData();
      formData.append('readfile', file);
      formData.append('fileName', file.name);
      const config = {
        headers: {'content-type': 'multipart/form-data',},
      };
      console.log (" filename => ", file.name);
      // Without axios file do not update. Browser does not pass data
      await axios.post(url, formData, config).then((response) => {
        let data = response.data;
        console.log("edit.js file loaded to server ??? " , data);
      });

    console.log ("submit create called, file upload over");
    // When a post request is sent to the create url, we'll add a new record
to the database.
    const newPerson = { ...form };

    await fetch("http://localhost:5000/record/add", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(newPerson),
    })
    .catch(error => {
      window.alert(error);
      return;
    });

    setForm({ name: "", position: "", level: "" });
    navigate("/");
  }

  function handleChange (e)
  {
        console.log (e);
        setFile(e.target.files[0] );
  }

   const handleFileUpload = () =>
    {
      console.log (" ==> file for upload ", file);
      //if (!file) return;
      const url = 'http://localhost:5000/upload';
      const formData = new FormData();
      formData.append('readfile', file);
      formData.append('fileName', file.name);
      const config = {
        headers: {'content-type': 'multipart/form-data',},
      };
      console.log (" filename => ", file.name);
      // Without axios file do not update. Browser does not pass data
       axios.post(url, formData, config).then((response) => {
        let data = response.data;
        console.log("edit.js file loaded to server ??? " , data);
      });
    }

  // This following section will display the form that takes the input from
the user.
  return (
    <div>
      <h3>Create New Record</h3>
      <form onSubmit={onSubmit}>
        <div className="form-group">
          <label htmlFor="name">Name</label>
          <input
            type="text"
            className="form-control"
            id="name"
            value={form.name}
            onChange={(e) => updateForm({ name: e.target.value })}
          />
        </div>
        <div className="form-group">
          <label htmlFor="position">Position</label>
          <input
            type="text"
            className="form-control"
            id="position"
            value={form.position}
            onChange={(e) => updateForm({ position: e.target.value })}
          />
        </div>
        <div className="form-group">
          <div className="form-check form-check-inline">
            <input
              className="form-check-input"
              type="radio"
              name="positionOptions"
              id="positionIntern"
              value="Intern"
              checked={form.level === "Intern"}
              onChange={(e) => updateForm({ level: e.target.value })}
            />
            <label htmlFor="positionIntern" className="form-check-
label">Intern</label>
          </div>
          <div className="form-check form-check-inline">
            <input
              className="form-check-input"
              type="radio"
              name="positionOptions"
              id="positionJunior"
              value="Junior"
              checked={form.level === "Junior"}
              onChange={(e) => updateForm({ level: e.target.value })}
            />
            <label htmlFor="positionJunior" className="form-check-
label">Junior</label>
          </div>
          <div className="form-check form-check-inline">
            <input
              className="form-check-input"
              type="radio"
              name="positionOptions"
              id="positionSenior"
              value="Senior"
              checked={form.level === "Senior"}
              onChange={(e) => updateForm({ level: e.target.value })}
            />
            <label htmlFor="positionSenior" className="form-check-
label">Senior</label>
          </div>
        </div>

        <div className="form-group">
        <label htmlFor="imageFile">ImageFile:</label>
        <input
            type="file"
            className="form-control"
            onChange={handleChange} // first file
        />
        <div>FILE INFO {file && file.name} </div>
        </div>
        <br/>

        <div className="form-group">
          <input
            type="submit"
            value="Create person"
            className="btn btn-primary"
          />
        </div>
```

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

Edit.js

```
import React, { useState, useEffect } from "react";
import { useParams, useNavigate } from "react-router";
import axios from 'axios';

export default function Edit() {
  const [form, setForm] = useState({
    name: "",
    position: "",
    level: "",
    records: [],
  });  // for text data entry

  const [file, setFile] = useState(); // holder for file

  const params = useParams();
  const navigate = useNavigate();
  console.log ("edit.js before useEffect ")

  useEffect(() => {
    async function fetchData() {
      const id = params.id.toString();
      const response = await
fetch(`http://localhost:5000/record/${params.id.toString()}`);

      if (!response.ok) {
        const message = `An error has occured: ${response.statusText}`;
        window.alert(message);
        return;
      }

      const record = await response.json();
      //console.log (" edit.js record useEffect fetchData () ", record);
      if (!record) {
        window.alert(`Record with id ${id} not found`);
        navigate("/");
        return;
      }

      setForm(record);
    }

    //console.log ('edit.js before call fetchData');
    fetchData();

    return;
  }, [params.id, navigate]);

  // These methods will update the state properties.
  function updateForm(value) {
    return setForm((prev) => {
      return { ...prev, ...value };
    });
  }

  async function onSubmit(e) {

    e.preventDefault();
    // here upload the file
    console.log ("submit called 0 ", file);
    if (file) {
        const url = 'http://localhost:5000/upload';
        const formData = new FormData();
        formData.append('readfile', file);
        formData.append('fileName', file.name);
        const config = {
          headers: {'content-type': 'multipart/form-data',},
        };
        console.log (" filename => ", file.name);
        // Without axios file do not update. Browser does not pass data
        await axios.post(url, formData, config).then( (response) => {
          let data = response.data;
          console.log("edit.js file loaded to server ??? " , data);
        });
    }

    const editedPerson = {
      name: form.name,
      position: form.position,
      level: form.level,
    };

    // This will send a post request to update the data in the database.
    await fetch(`http://localhost:5000/update/${params.id}`, {
      method: "POST",
      body: JSON.stringify(editedPerson),
      headers: {
        'Content-Type': 'application/json'
      },
    });

    console.log ("submit called.. over ");
    navigate("/");
  }

  function handleChange (e)
  {
        console.log (e);
        setFile(e.target.files[0] );
  }

   const handleFileUpload = () =>
    {
      console.log (" ==> file for upload ", file);

      if (!file) return;

      //event.preventDefault()
      const url = 'http://localhost:5000/upload';
      const formData = new FormData();
      formData.append('readfile', file);
      formData.append('fileName', file.name);
      const config = {
        headers: {'content-type': 'multipart/form-data',},
      };
      console.log (" filename => ", file.name);
      // Without axios file do not update. Browser does not pass data
      axios.post(url, formData, config).then( async (response) => {
        let data = await response.data;
        console.log("edit.js file loaded to server ??? " , data);
      });

    }

  // This following section will display the form that takes input from the
user to update the data.
  return (
    <div>
      <h3>Update Record</h3>
      <form onSubmit={onSubmit}>
        <div className="form-group">
          <label htmlFor="name">Name: </label>
          <input
            type="text"
            className="form-control"
            id="name"
            value={form.name}
            onChange={(e) => updateForm({ name: e.target.value })}
          />
        </div>
        <div className="form-group">
          <label htmlFor="position">Position: </label>
          <input
            type="text"
            className="form-control"
            id="position"
            value={form.position}
            onChange={(e) => updateForm({ position: e.target.value })}
          />
        </div>
        <div className="form-group">
          <div className="form-check form-check-inline">
            <input
              className="form-check-input"
              type="radio"
              name="positionOptions"
              id="positionIntern"
              value="Intern"
              checked={form.level === "Intern"}
              onChange={(e) => updateForm({ level: e.target.value })}
            />
            <label htmlFor="positionIntern" className="form-check-
label">Intern</label>
          </div>
          <div className="form-check form-check-inline">
            <input
              className="form-check-input"
              type="radio"
              name="positionOptions"
              id="positionJunior"
              value="Junior"
              checked={form.level === "Junior"}
              onChange={(e) => updateForm({ level: e.target.value })}
            />
            <label htmlFor="positionJunior" className="form-check-
label">Junior</label>
          </div>
          <div className="form-check form-check-inline">
            <input
              className="form-check-input"
              type="radio"
              name="positionOptions"
              id="positionSenior"
              value="Senior"
              checked={form.level === "Senior"}
              onChange={(e) => updateForm({ level: e.target.value })}
            />
            <label htmlFor="positionSenior" className="form-check-
label">Senior</label>
        </div>
        </div>

        <br />
        <div className="form-group">
        <label htmlFor="imageFile">ImageFile:</label>
        <input
            type="file"
            className="form-control"
            onChange={handleChange} // first file
        />
        <div>FILE INFO {file && file.name} </div>
        </div>
        <br/>

        <div className="form-group">
          <input
            type="submit"
            value="Update Record"
            className="btn btn-primary"
          />
        </div>
      </form>
    </div>
  );
}

import React, { useEffect, useState } from "react";
import { Link } from "react-router-dom";

//import myImage1 from "../bahadur.png"
//var photo = require=(`${props.record.imgPath}`);
// Import Logo from "./Logo.svg"

const Record = (props) => (
  <tr>
    <td>{props.record.name}</td>
    <td>{props.record.position}</td>
    <td>{props.record.level}</td>
    <td>
      <Link className="btn btn-link"
to={`/edit/${props.record._id}`}>Edit</Link> |
      <button className="btn btn-link"
        onClick={() => {
          props.deleteRecord(props.record._id);
        }}
      >
        Delete
      </button>
    </td>
     <td><img src={`data:image/png;base64,${props.record.imgDataStr}`}
alt='no image' width='50' height='40' ></img> </td>

  </tr>
);

export default function RecordList() {
  const [records, setRecords] = useState([]);

  // This method fetches the records from the database.
  useEffect(() => {
    async function getRecords() {
      const response = await fetch(`http://localhost:5000/record/`);

      if (!response.ok) {
        const message = `An error occured: ${response.statusText}`;
        window.alert(message);
        return;
      }

      const records = await response.json();
      console.log ("useEffet() records", records);
      setRecords(records);
    }

    getRecords();

    return;
  }, [records.length]);

  // This method will delete a record
  async function deleteRecord(id) {
    await fetch(`http://localhost:5000/${id}`, {
      method: "DELETE"
    });

    const newRecords = records.filter((el) => el._id !== id);
    setRecords(newRecords);
  }

  // This method will map out the records on the table
  function recordList() {
    return records.map((record) => {
      return (
        <Record
          record={record}
          deleteRecord={() => deleteRecord(record._id)}
          key={record._id}
        />
      );
    });
  }

  // This following section will display the table with the records of
individuals.
  return (
    <div>
      <h3>Record List got from MongoDB (records) </h3>
      <table className="table table-striped" style={{ marginTop: 20 }}>
        <thead>
          <tr>
            <th>Name</th>
            <th>Position</th>
            <th>Level</th>
            <th>Action</th>
            <th>Image</th>
          </tr>
        </thead>
        <tbody>{recordList()}</tbody>
      </table>
    </div>
  );
}
```

Navbar.js

```
import React from "react";

// We import bootstrap to make our application look better.
import "bootstrap/dist/css/bootstrap.css";

// We import NavLink to utilize the react router.
import { NavLink } from "react-router-dom";

// Here, we display our Navbar, PNG is loaded local direct using require
export default function Navbar() {
  return (
    <div>
      <nav className="navbar navbar-expand-lg navbar-light bg-light">
        <NavLink className="navbar-brand" to="/">
        <img style={{"width" : 25 + '%'}}
src={require=('./Mongo_DB.png')}></img>
        </NavLink>
        <button
          className="navbar-toggler"
          type="button"
          data-toggle="collapse"
          data-target="#navbarSupportedContent"
          aria-controls="navbarSupportedContent"
          aria-expanded="false"
          aria-label="Toggle navigation"
        >
          <span className="navbar-toggler-icon"></span>
        </button>

        <div className="collapse navbar-collapse"
id="navbarSupportedContent">
          <ul className="navbar-nav ml-auto">
            <li className="nav-item">
              <NavLink className="nav-link" to="/create">
                Create Record
              </NavLink>
            </li>
            <li className="nav-item">
```

```
                <NavLink className="nav-link" to="/upload">
                  Image File Upload
                </NavLink>
              </li>
            </ul>
          </div>
        </nav>
      </div>
    );
}
```

Fileupload.js

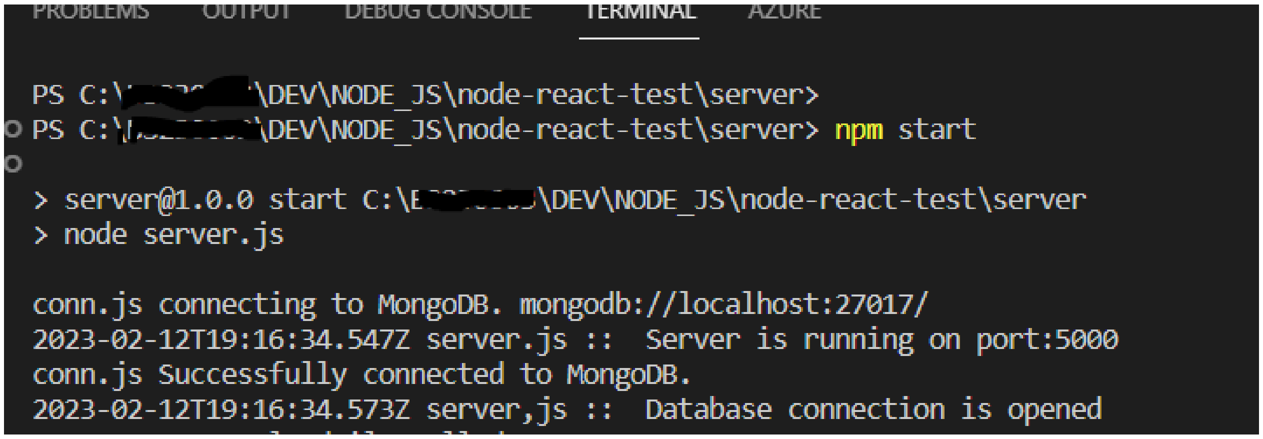
```
import React, { useEffect, useState } from "react";
import axios from 'axios';

export default function UploadFile (event) {
    const [file, setFile] = useState(); // holder for file

    const handleFileUpload = () =>
    {
      console.log (" ==> file for upload ", file);

      if (!file) return;

      //event.preventDefault()
      const url = 'http://localhost:5000/upload';
      const formData = new FormData();
      formData.append('readfile', file);
      formData.append('fileName', file.name);
      const config = {
        headers: {
          'content-type': 'multipart/form-data',
        },
      };
      console.log (" data=> ", file, file.name);
      // Without axios file do not update. Browser does not pass data
      axios.post(url, formData, config).then( async (response) => {
        let data = await response.data;
        console.log("file loaded to server ??? " , data);
      });

    }

    function handleChange (e)
    {
        console.log (e);
        setFile(e.target.files[0] );
    }

    return (
    <form onSubmit={handleFileUpload}>
        <div>
        <label htmlFor="imageFile">ImageFile:</label>
        <input
            type="file"
            className="form-control"
            onChange={handleChange} // first file
        />
        <div>FILE INFO {file && file.name} </div>
        <button
            type="submit"
            className="btn btn-primary">Upload File</button>
        </div>
    </form>
    );
}
```

Copy-Paste code in your environment. Make sure MongoDB database setup installation runs on default localhost, it has no user password defined. Just an example to know MERN and CRUD

To run this Program

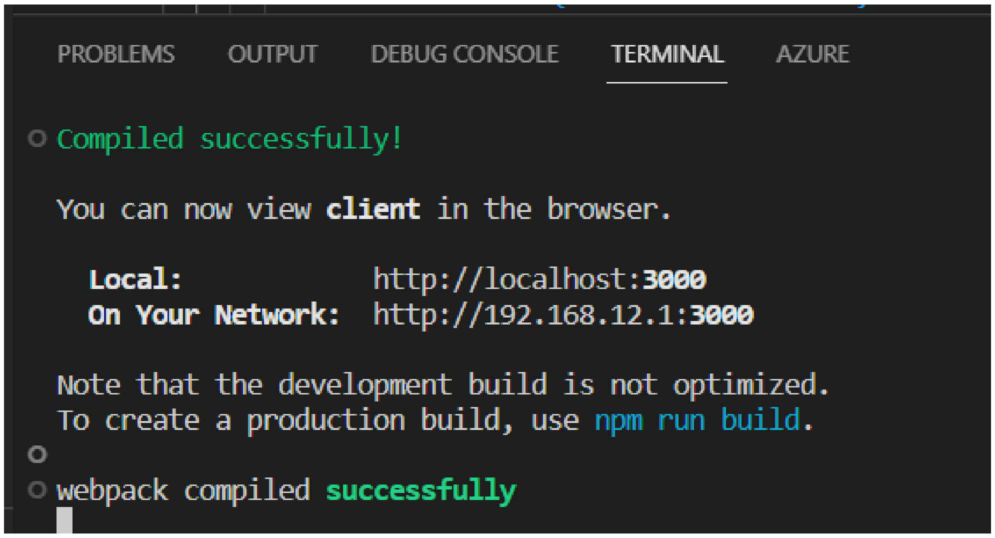
you need two Terminals inside the Visual Studio Code,

Server Side Terminal:



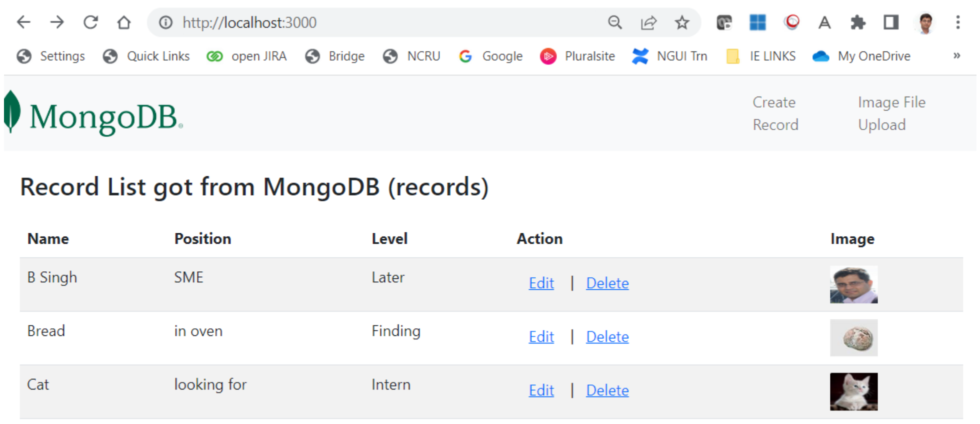
```
PS C:\[redacted]\DEV\NODE_JS\node-react-test\server>
PS C:\[redacted]\DEV\NODE_JS\node-react-test\server> npm start

> server@1.0.0 start C:\[redacted]\DEV\NODE_JS\node-react-test\server
> node server.js

conn.js connecting to MongoDB. mongodb://localhost:27017/
2023-02-12T19:16:34.547Z server.js ::  Server is running on port:5000
conn.js Successfully connected to MongoDB.
2023-02-12T19:16:34.573Z server,js ::  Database connection is opened
```

Client Side Terminal:



```
Compiled successfully!

You can now view client in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.12.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

It will open the Browser with this Window.



Record List got from MongoDB (records)

Name	Position	Level	Action	Image
B Singh	SME	Later	Edit | Delete	
Bread	in oven	Finding	Edit | Delete	
Cat	looking for	Intern	Edit | Delete	

Created by Bahadur Singh: singh.bahadur@gmail.com works at NCR Corp as

SME / Project Manager. Leads the Software Development for the customer based projects.