Since @Adam asked nicely 🙂...

**1. OVERVIEW**

For some context, npm start fires a command that creates and runs a **development** web server that's used to serve up the React App via localhost:3000 by default. Take note that it's a **development** server therefore you wouldn't use npm start to deploy the app in production.

This MERN example is a simple React App (using Functional Components) with a Express + MongoDB backend. We will add, save and display two pieces of information, the Forename and Age of People.

**React**- [http://localhost:3000](http://localhost:3000/)

**Express**- [http://localhost:4747](http://localhost:4747/)

**MongoDB**- mongodb://localhost:27017/reactDB

Caveat: I assume that anyone reading this has completed the course, can get a mongod instance up and running, and can confidently reference documentation to understand all the packages used.

Let's get started...

**2. SETUP FOLDERS**

1. mkdir react-with-backend // Root Project folder
2. cd react-with-backend // Make sure to cd into it
3. mkdir client // Frontend folder for React App
4. mkdir server // Backend folder for Express Server + MongoDB

**3. SETUP EXPRESS AND MONGODB (BACKEND)**

  a) cd server

  b) Create a **server.js** file

  c) Run npm init and set the Entry Point to **server.js** (not index.js)

  d) npm install express mongoose cors dotenv

  e) Copy-Paste-Save the following code into the **server/server.js** file

1. require("dotenv").config();
2. const express  = require("express");
3. const mongoose = require("mongoose");
4. // const cors     = require("cors");
5. const path     = require("path");
6. const app      = express();
8. const PORT     = process.env.PORT || 4747;
9. const DB\_URI   = "mongodb://localhost:27017/"
10. const DB       = "reactDB";
12. // Middleware
13. app.use(express.json());
14. app.use(express.urlencoded({ extended: true }));
15. // app.use(cors());
17. // Establish DB connection
18. mongoose.connect(DB\_URI + DB, {
19. useUnifiedTopology: true,
20. useNewUrlParser: true,
21. useCreateIndex: true,
22. useFindAndModify: false,
23. connectTimeoutMS: 10000
24. });
26. const db = mongoose.connection;
28. // Event listeners
29. db.once('open', () => console.log(`Connected to ${DB} database`));
31. // Create Schema
32. let PersonSchema = new mongoose.Schema(
33. {
34. forename: String,
35. age: Number
36. },
37. { collection: "people" }
38. );
40. // Create Model
41. let PersonModel = db.model("PersonModel", PersonSchema);
43. // Route to Get all People
44. app.get("/api/people", (req, res) => {
45. PersonModel.find({}, {\_\_v: 0}, (err, docs) => {
46. if (!err) {
47. res.json(docs);
48. } else {
49. res.status(400).json({"error": err});
50. }
51. });
52. })
54. // Route to Add a Person
55. app.post("/api/person/add", (req, res) => {
56. let person = new PersonModel(req.body);
58. person.save((err, result) => {
59. if (!err) {
60. delete result.\_doc.\_\_v;
61. res.json(result.\_doc);
62. } else {
63. res.status(400).json({"error": err});
64. }
65. });
66. })
68. app.listen(PORT, () => {
69. console.log(app.get("env").toUpperCase() + " Server started on port " + (PORT));
70. });

**4. SETUP REACT APP (FRONTEND)**

  a) cd ..  I.e. cd to the "react-with-backend" folder

  b) npx create-react-app client

  c) npm install react react-dom react-scripts axios

  d) Add the following entry to package.json which will act as the proxy to the server

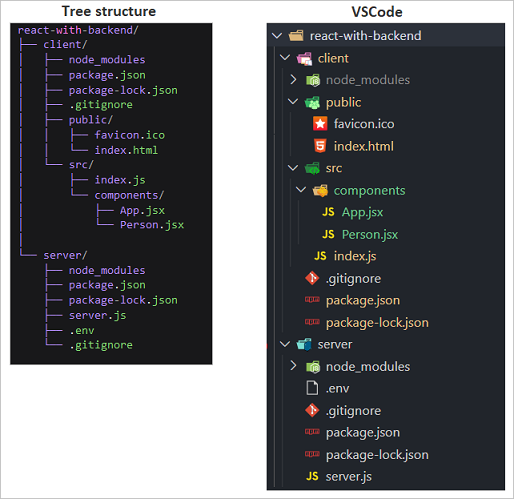
1. "proxy": "http://localhost:4747",

   It should look like this:



   The proxy will enable us make a GET/POST request like "/api/people" instead of writing "[http://localhost:4747/api/people"](http://localhost:4747/api/people) and ensures we don't get a CORS error. This proxy setting only works in development.

  e) Delete all irrelevant files by making your project folder look like this:



  f) Create the following files and their scripts:

**client/public/index.html**

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="utf-8" />
5. <link rel="icon" href="%PUBLIC\_URL%/favicon.ico" />
6. <meta name="viewport" content="width=device-width, initial-scale=1" />
7. <title>MERN example</title>
8. </head>
9. <body>
10. <noscript>You need to enable JavaScript to run this app.</noscript>
11. <div id="root"></div>
12. </body>
13. </html>

**client/src/components/Person.jsx**

1. import React from "react";
3. function Person(props) {
4. const { forename, age } = props;
6. return (
7. <>
8. <h3>{forename + " is " + age + " years old"}</h3>
9. </>
10. );
11. }
13. export default Person;

**client/src/components/App.jsx**

1. import React, { useState, useEffect } from "react";
2. import Person from "./Person";
3. import axios from "axios";
5. /\*\*\* Remember that the proxy to the backend server is http://localhost:4747 \*\*\*/
7. const defaultPerson = {
8. forename: "",
9. age: "",
10. };
12. function App() {
13. const [person, setPerson] = useState(defaultPerson);
14. const [people, setPeople] = useState([]);
16. /\*
17. Fetch all People the moment this App component loads for the first time
18. Notes: The proxy enables us to use axios without the full url http://localhost:4747/api/people
19. The empty array [] parameter ensures that the code inside useEffect() runs once
20. \*/
21. useEffect(() => {
22. axios.get("/api/people")
23. .then((res) => setPeople(res.data))
24. .catch((err) => console.error(err));
25. }, []);
27. function handleChange(event) {
28. const { name, value } = event.target;
30. setPerson({ ...person, [name]: value });
31. }
33. /\*
34. Add a person to DB and update state
35. Notes: The proxy enables us to use axios without the full url http://localhost:4747/api/person/add
36. \*/
37. function addPerson(newPerson) {
38. axios.post("/api/person/add", newPerson)
39. .then((res) => setPeople([...people, res.data]))
40. .catch((err) => console.log(err));
41. }
43. return (
44. <>
45. <form onSubmit={(e) => e.preventDefault()}>
46. <input
47. name="forename"
48. type="text"
49. placeholder="Enter your Forename"
50. value={person.forename}
51. onChange={handleChange}
52. />
53. <input
54. name="age"
55. type="text"
56. placeholder="Age"
57. value={person.age}
58. onChange={handleChange}
59. />
60. <button
61. onClick={() => {
62. if (person.forename && person.age) {
63. addPerson(person);
64. setPerson(defaultPerson);
65. }
66. }}
67. >
68. Add
69. </button>
70. {people.map((person) => (
71. <Person
72. key     ={person.\_id}
73. \_id     ={person.\_id}
74. forename={person.forename}
75. age     ={person.age}
76. />
77. ))}
78. </form>
79. </>
80. );
81. }
83. export default App;

**client/src/index.js**

1. import React from "react";
2. import ReactDOM from "react-dom";
3. import App from "./components/App";
5. ReactDOM.render(<App />, document.getElementById("root"));

**5. LET'S GET RUNNING...**

  a) Open two terminals, one for the server and the other for the client

  b) Terminal 1, launch the backend server first:

     cd server

     \*\*\* Reminder! Make sure port 4747 is free \*\*\*

     \*\*\* Reminder! Make sure mongod is running \*\*\*

     nodemon server.js

  c) Terminal 2, launch the React App:

     cd client

     \*\*\* Reminder! Make sure port 3000 is free \*\*\*

     npm start

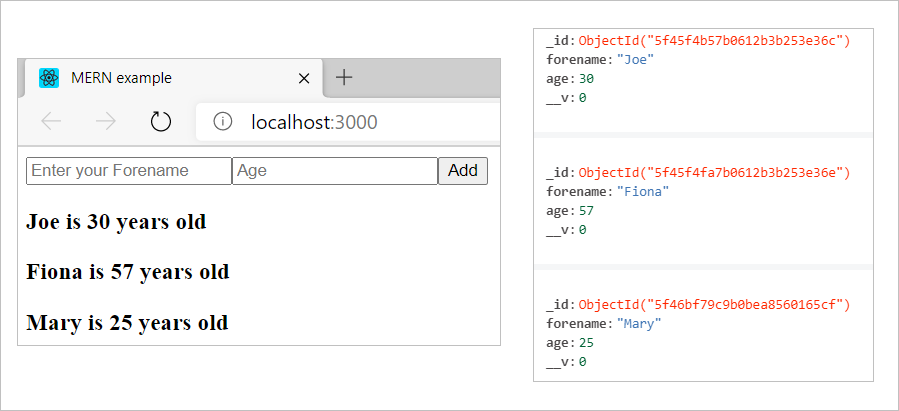
  d) Step (c) should open a browser. Otherwise, open a browser and enter [http://localhost:3000](http://localhost:3000/)

  e) Add a couple of Names and Ages in the app and look in a mongo shell to see the results:

1. mongo
2. use reactDB
3. db.people.find().pretty()

  f) Refresh the browser (or open a new one) and all the Name/Age documents will be retrieved from the db and displayed

**6. RESULT**



**7. TROUBLESHOOTING AND TIPS**

  - If you see a CORS related error, first make sure that you completed step 4d correctly. As a last resort, comment out the two "cors" lines of code in server.js, save and retry.

  - *"Proxy error: Could not proxy request /api/people from localhost:3000 to*[*http://localhost:4747*](http://localhost:4747/)*/"*. The port 4747 is most likely used by another process. Change the proxy port in step 4d and in server.js to something else (e.g. 4000), save the files, stop and re-run npm start (don't rely on the auto-restart).

  - There's an npm package called concurrently that allows you to start both client and server using one command. They'll still run on different servers, but it'll run both using a single command.

**8. FINAL NOTES**

The .env and .gitignore files are not mandatory for this exercise but recommended for obvious reasons.

In a **Production**build, you can serve the React App from the Express server that MongoDB is running on, i.e. from the same domain and port, or have them separate. I've not included these steps but let me know if you're unable to figure it out.