ReadMe

Sunday, 4 October 2020 11:29 PM

Tech Stack used:
Front end: React, Bootstrap
Back end: Node.js (express framework)
DB: MongoDB

1. Created a Mongo Atlas account - sm2322 @scarletmail.rutgers.edu
2. Created a cluster 512 MB free forever - GCP
3. We will revisit on how to config this Mongo thingy later
    a. It will then ask for whitelist IP address to access the mongoDB cluster - say current ip
    b. Choose a atlasAdmin password
    c. Connection method - connect our application to cluster using mongoDB's native drivers
4. Check if node is installed ? Command: node -v
5. Using npx - node package execute to create a react project using create react app without installing react itself :P
6. npx create-react-app mern-exercise-tracker
    a. it creates a directoy containing default react project template with all dependencies
7. Explained MongoDB ObjectID - autoamtically created by Mongo driver - Unique for each object across the collection - while we can define our own ObjectIds - we are letting Mongo driver to do that for us in this proj

8. go to mern directory
9. normally, if we want to start web de v server. we. type npm start but first we are gonna create a backend and then front end.
10. Plan is to create backend -> connecting it to mongo db atlas in google cloud -> react frond end
11. mkdir backend -> cd backend
12. create a package.json file by using npm init -y
    a. -y tells yes to everything
13. Install few dependencies
    a. npm install express cors mongoose dotenv
        i. express -> lightweight fast web framework for node js
        ii. cors -> cross origin resource sharing - this allows ajax to skip same origin policy - and access resources from remote hosts
            1) it also provides an express middleware that enables cors with different options - which eases our access of information for our server from outside the server
        iii. Mongoose -> makes interacting with mongo db thriugh node js simpler
        iv. dotenv - which loads env variables from dotenv file into process.env
            1) this makes dev simpler - instead setting env variables in dev machine we can put them in a file - we will dotenv later!!
14. npm install -g nodemon
    a. Helps dev easier - auto restart node app i:e server when it sees file changes in directory (god sent tool!!!)

(god sent tool!!!)

15. Time to create the server babe!!!
    a. create a server.js file in backend dir
    b. Include the packages needed - express, cors, mongoose
    c. check the file for the code
16. nodemon server -> will start the server
17. Now, we are at the point that we can connect to our database on mongodb atlas
18. BTW - mongo db project MERN has the atlasAdmin username/password - harsha_musunuri/spy4d@12
19. Put the URI we get in mongoDB atlas - connect to cluster() page into .env as the environmental variable
20. Put things in DB and read from the DB
21. Create schema for the DB using Mongoose
    a. We have two in these 1. execises and 2. the users
22. Create a new folder named models in backend
    a. create two files
        i. exercise.model.js
        ii. user.model.js
23. Check the .js files for both the models
24. Add the Api endpoint routes - so the server can be used to perform CRUD (create, read, update, delete) operations on the DB
25. mkdir in backend -> routes
26. Add two files in it
    a. exercises.js
    b. users.js
27. Tell the server that we created these two files by requiring them in the mummy file aka. server.js
28. Build the exercises.js and users.js

28. ~~Build the exercises.js and users.js~~
29. We built the get and post requests on the respective routes in routes/exercises.js , routes/users.js
30. Lets test the server API - we can use postman or Insomnia to test this !! Haha what a name man!!
31. Downloading Insomnia Core and Installing
32. After installation - we did a post request at the URL: http://localhost:5000/users/add - couple of users, so we can test get method later
33. We tested the get method - got both the users I added, with the mongoID - remember we were talking about this mongoID :P
34. I saw that the same values we posted are in MongoDB Atlas as well - haha, this felt good!
35. Posting couple of Exercises as well
36. Getting them back - saw timestamps as well - createdAt and modifiedAt
37. Also, we can just update the database itself, and by doing get request - the updated values will be shown
38. We are gonna have create and read operations for Users but Exercises we gotta have all the CRUD operations on the frontend of our application.
39. Added the Update and Delete Api endpoints and checked them
40. One thing observed is to update just the description of the exercise, we got to send all the schema instead of just the description - This we might solve later!!! Let's see
41. Done with the Backend!!!!!!!!!
42. React babe!!
43. Flexible JS libraries for building UI
    a. lets us compose complex user interfaces from

small and isolated pieces of code called components
    b. We use components to tell react what to be shown on the screen
    c. Components take arguments called props and returns a hierarchy of views to display through the render method
    d. syntax may look like html but its called JSX
44. Remember the react app we created at the very beginning ? (check point 5). We are heading back there now
45. In the created React App directory /public/index.html -> this html is what we see as our web app
46. Coding App.js -> what that displays on the front end
47. cd mern-exercise-tracker -> npm start
    a. this started the react app - it showed the react logo, much better than a simple html css thingy
    b. Also, if we do some changes to App.js the page reloads automatically
48. The flow is this App.js is the react App -> loaded by index.js to index.html on a div element
49. Lets start our App, starting with some bootstrap.css styling
50. install bootstrap -> npm install bootstrap
51. Import bootstrap in our App.js -> import "bootstrap/dist/css/bootstrap.min.css"
52. Setup React router
    a. npm install react-router-dom
    b. It is helpful in routing different URL to differnet React components - which we will see in a while!!
    c. import {BrowserRouter as Router, Route} from

    c. import {BrowserRouter as Router, Route} from "react-router-dom"
    d. Whatever we need to use with Router("imported from react-router-dom") we keep it in the <Router></Router> element
    e. Inside the Router element - we have Route elements that helps in choosing a react component based on the path
53. Import the component files in the app.js (these component files are yet to be written)
54. Create a components directory in the same directory as app.js
    a. add navbar.component.js
    b. add edit-exercises.component.js
    c. add create-exercise.compnent.js
    d. add create-user.component.js
    e. add exercises-list.component.js
55. Installed Visual Studio - its very good btw
    a. view -> terminal will show the terminal in the editor itself
56. Check the code files for those compoenents.
57. npm start -> showed the website its live
58. In React, state is called the variable whenever we update the state -> it updates the look of the website
59. Whenever constructor is called with props, super is also called with props
60. We added couple of methods in create-exercise.component.js that helps in adding the exercise
61. Created a Form, for the date picking we need npm install react-datepicker
62. Similarly, wrote the create-user.component.js

63. Connecting front end with back end - causing front end send http requests to the server at the backend - we use axios library to do that - npm install axios (Remember how we used Insomnia to send http requests to the server and updated the DB ? We do the same but insomnia role wil be taken by the front end)
64. axios.post is a post req to the server -> server takes the values in request body and does the backend updates - witnessed that
65. edit-exercises.component.js.exercises-list.component.js is done. The app is fully functional now