In this section we are going to multi containers, how they are going to setup the multiple services front end and backend on different containers with some realistic Deployment of apps on multiple containers.

Working with Multiple Containers

Combining Multiple Services to One App

We are going the following project, we will three main building blocks.

1. Database 🡪 MongoDB (To store the data which was generated by application)
2. Backend server 🡪 NodeJS REST API (It is just NodeJS web application server which exposes some GUI, which doesn’t respond html data and simply accepts and returns json data). It is not exposed directly to user.
3. Frontend Server 🡪 React Single page application (This is a simple webpage which will accepts users Goals data and store it in Mongo DB). It is exposed directly to user at front end webpage.

This is pretty common type of setup in now a days, our backend will talk to database to fetch the data, and our front end will talk to backend server.

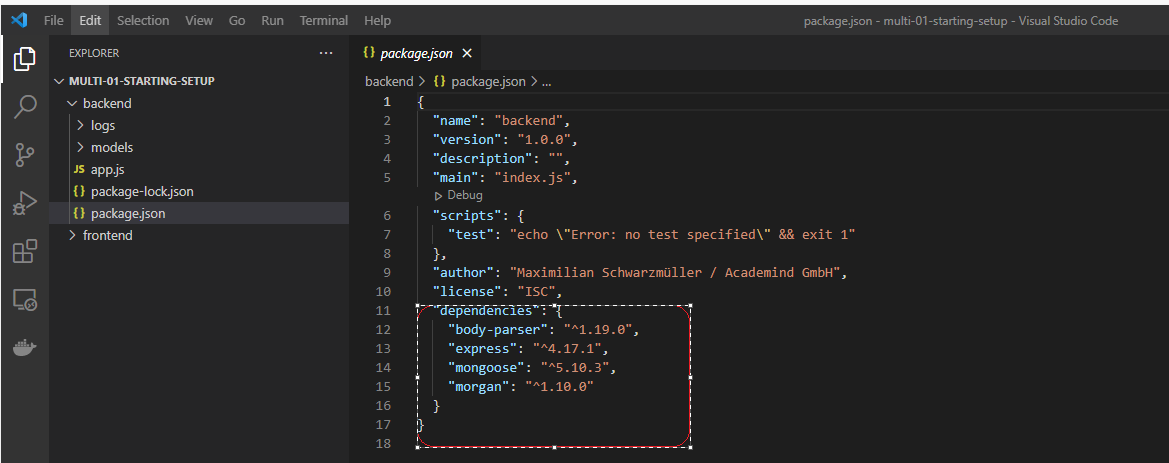
Download the application from the following GitHub URL

<https://github.com/harikrishna83/Docker-firstdemo-handson/tree/master/multi-01-starting-setup>

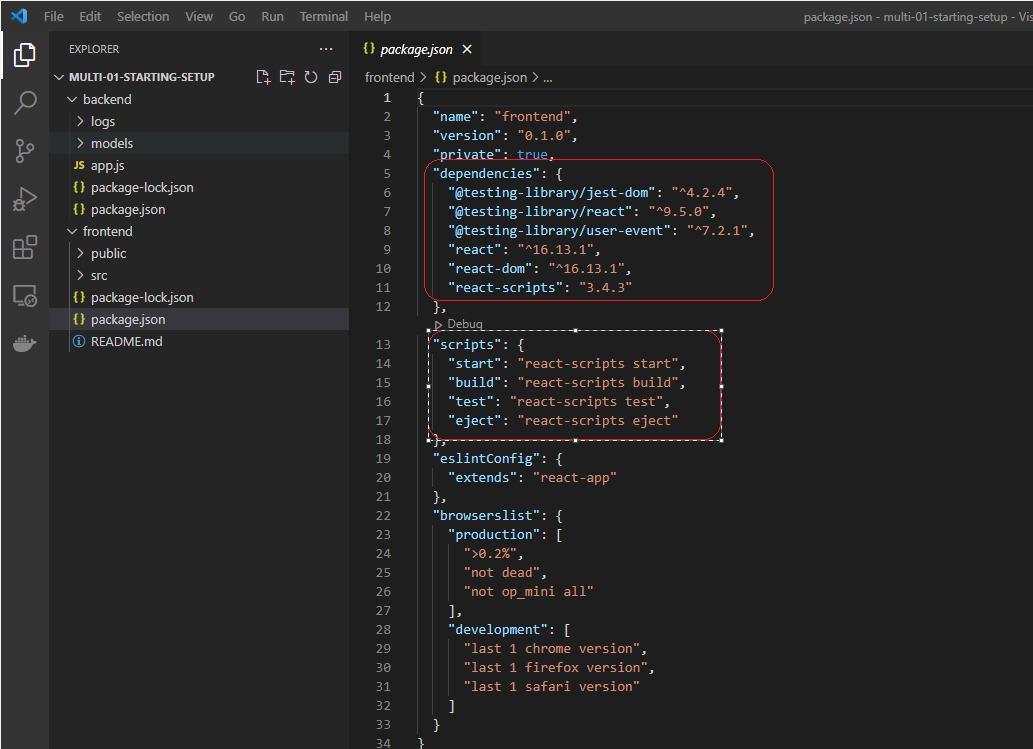
On backend server, it will accept some goals and store it on database and mainly on backend NodeJS server it will store the logs on logs folder.

On front end server, react is a java script library which runs on a browser not on a server, which will accept the goals from user interface. It is detached from the backend and a standalone server, I am just sending a request to a backend API <http://localhost/goals> to post the new goals or to get a already stored goals or to delete the goals using API’s on backend.

On backend the following are the dependencies need to be installed.



On frontend server, the following dependencies need to be installed and also the following scripts will take care of React process.



Backend server listens on port 80 and front end server listens on 3000.

Here is the final requirements on our project.

1. Data on Mongo DB need to be persist and access should be limited.
2. On Backend server, the logs files need to be persistent and Source code updates need to be on fly.
3. On Frontend server, source code updates need to be reflect on fly

Access should be limited

React SPA

MongoDB

NodeJS REST API

Three Building Blocks

Live Source Code Update

Live Source Code Update

Data must persist

Data must persist

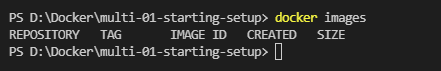
Frontend

Backend

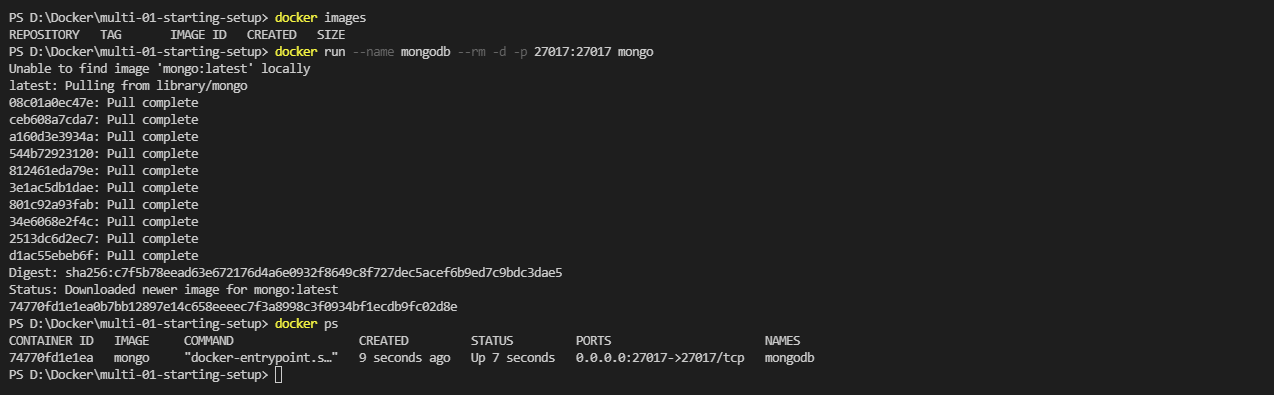
Database

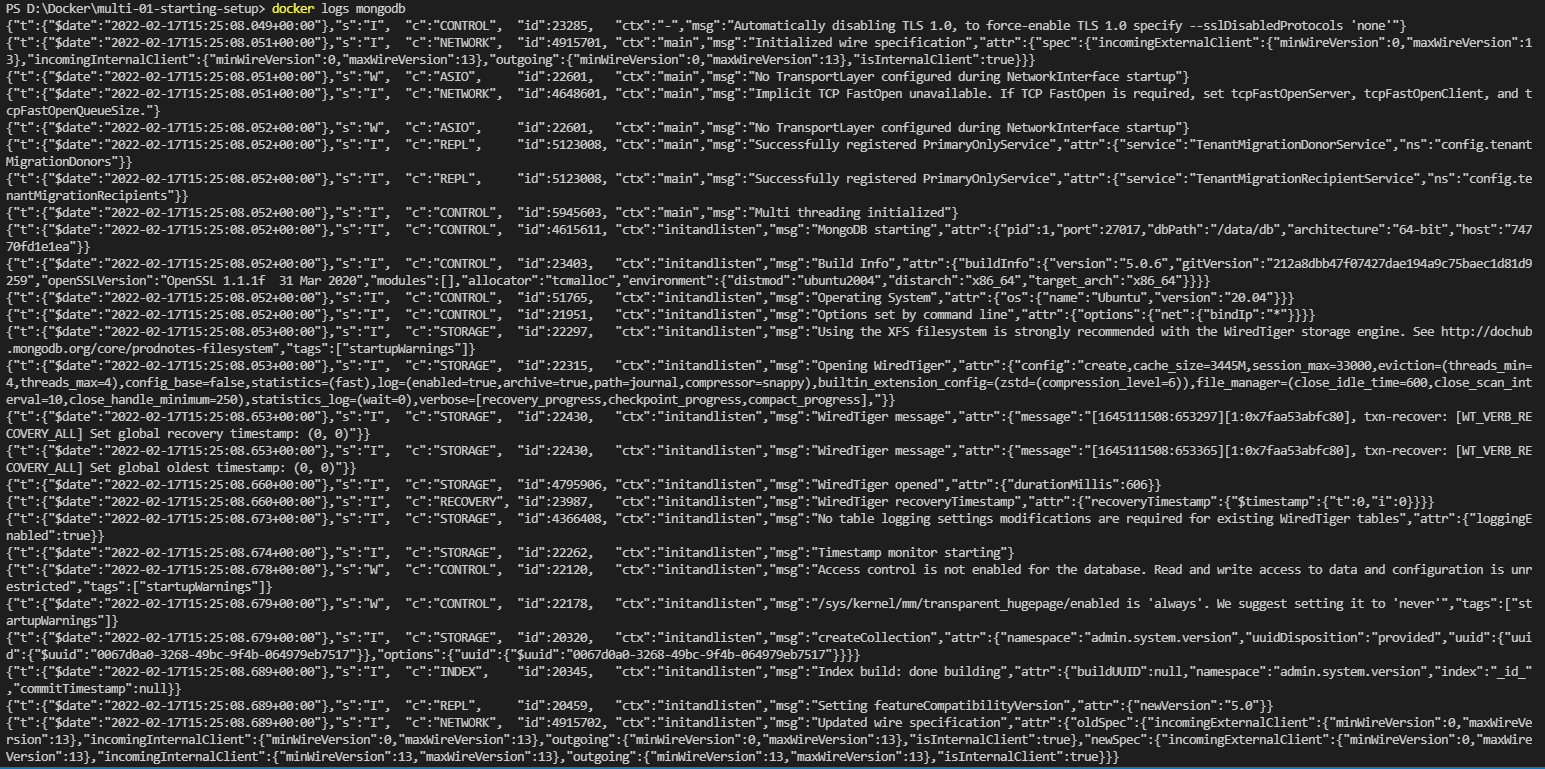
## Dockerizing Mongo DB

We need to dockerize the mongoDB by downloading official image from docker hub using the following command “docker run mongo”



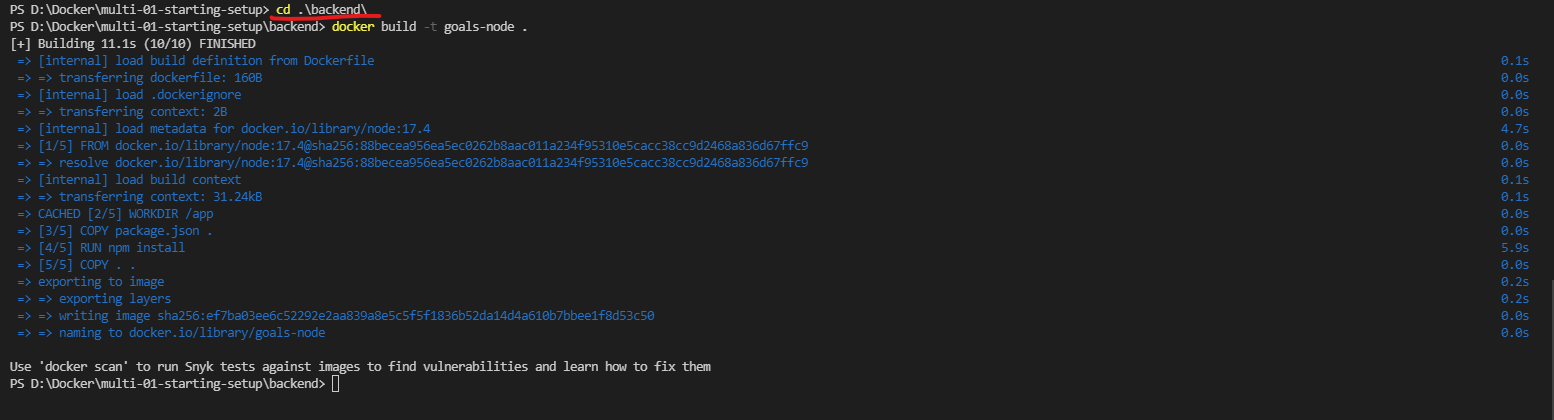
Suppose if your nodeJS and react are not containerized and are running on some servers in order to connect this mongo DB with NodeJS & React servers locally we need to expose the port using –p 27017:27017



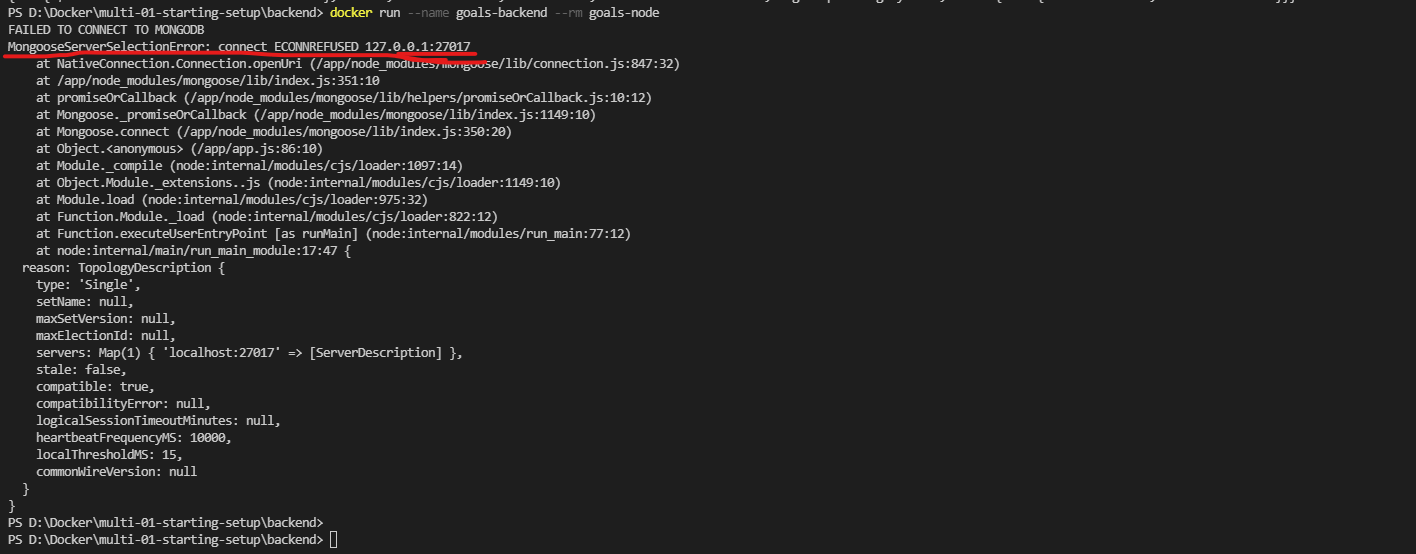


## Dockerizing the node app

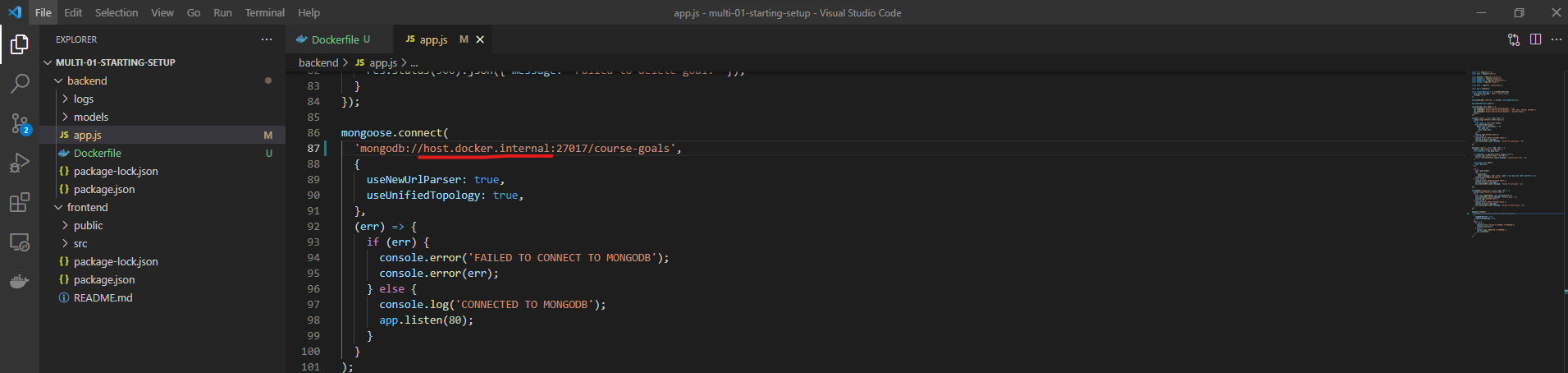
Now we need write the Dockerfile in order to start the Nodejs with our customizations, we didn’t add the Dcokerfile to mongodb as we had used the direct one and didn’t done any customizations on that.

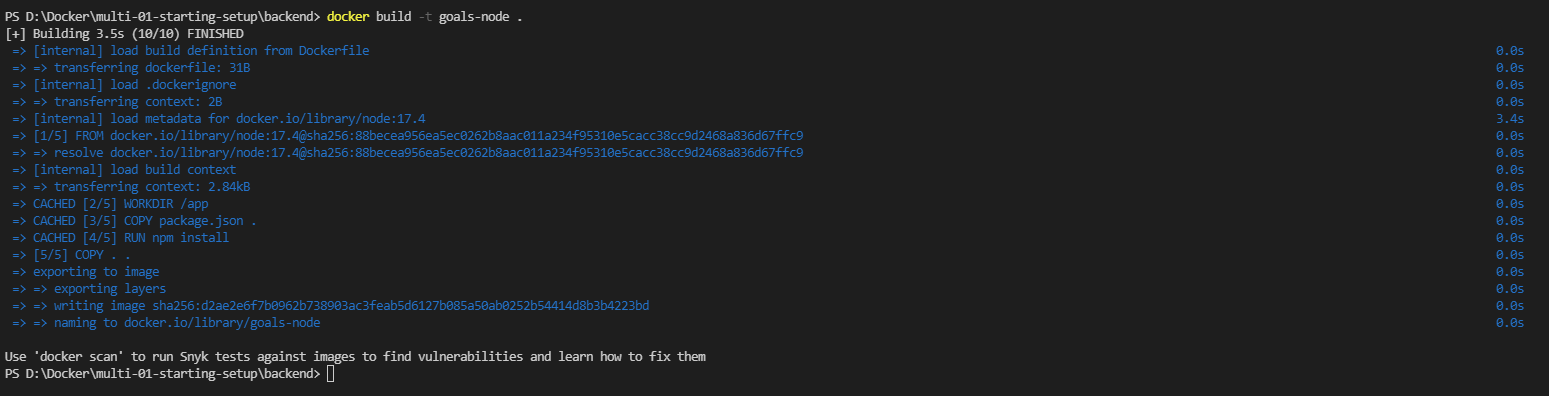


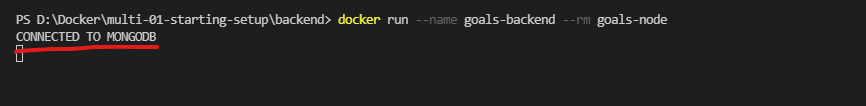
When I am trying to run the container, initially it will running but when the application to connect to the database it will fail it.



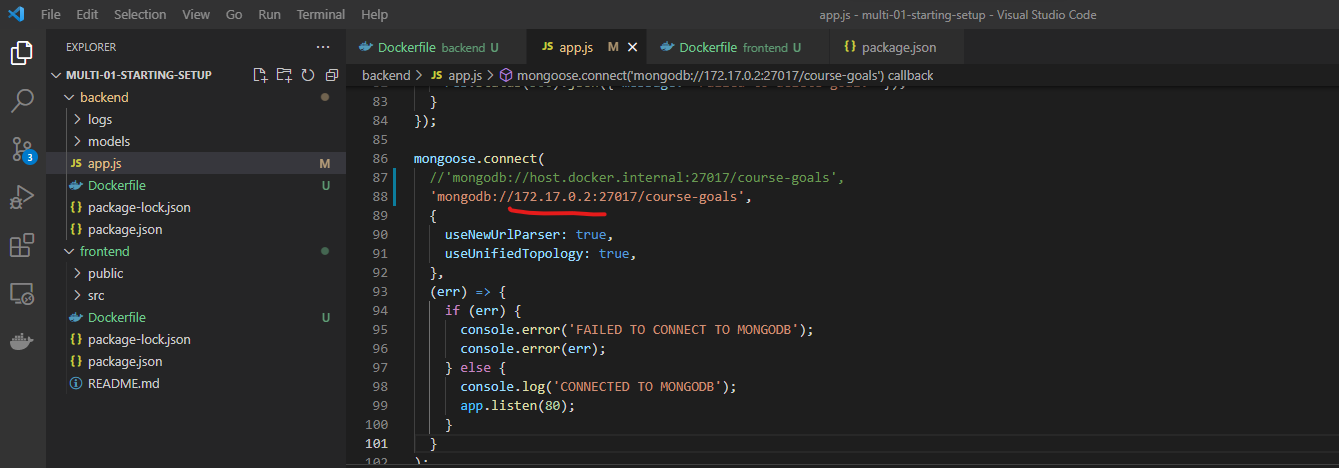
Here in the dockerize backend application I am trying to reach the mongoDB locally, as it was not installed on this container, it was separate container we need to solve this by changing the localhost to special domain “host.docker.internal”, which is translated to your real host machine IP by docker.

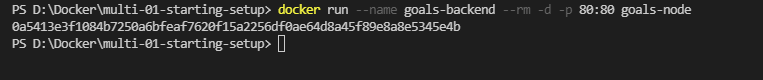


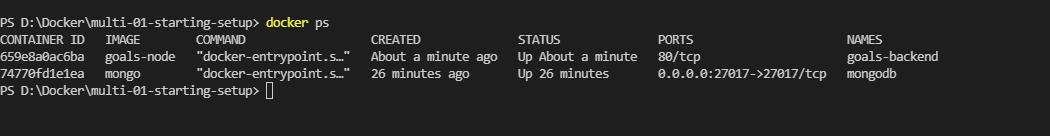




As I had used “host.docker.internal”, it got connected to mongodb on my localhost. If you want to connect to the mongodb on container, you need to replace the IP of the mongoDB container by using the command docker container inspect mongodb.

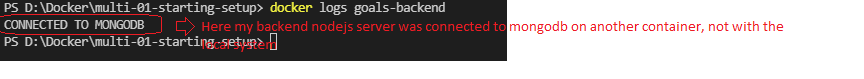


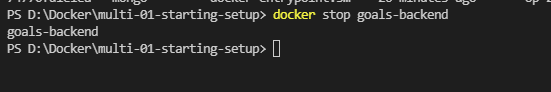


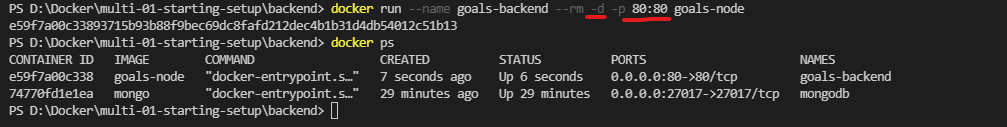


As we have front end server REACT to communicate to this backend nodejs, it is always suggestible to publish the port.

Note: If you want to communicate with the services on the containers, I mean not to a localhost or outside world you don’t need to publish the port.

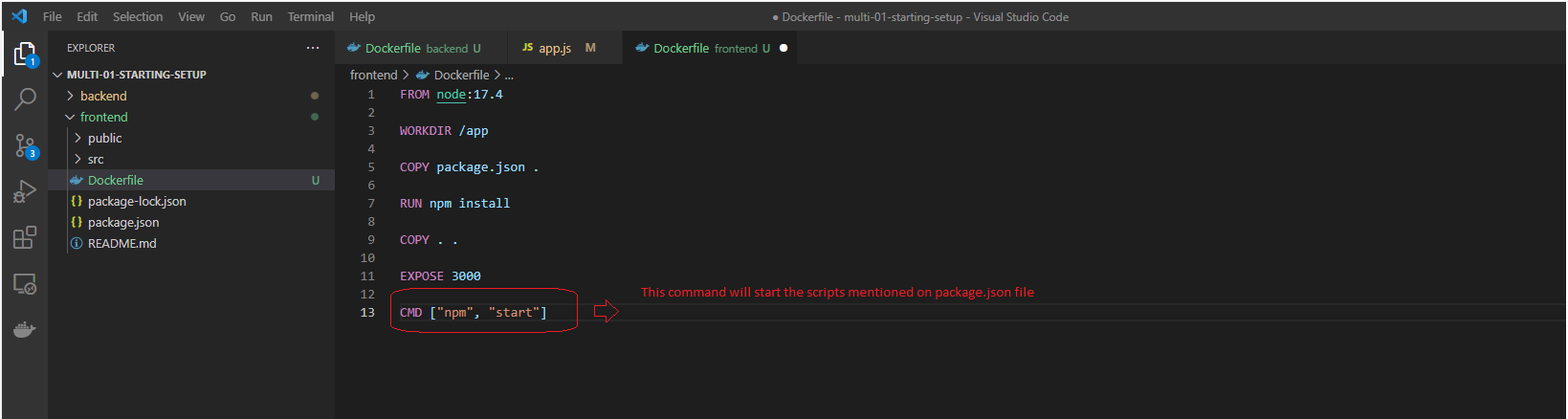


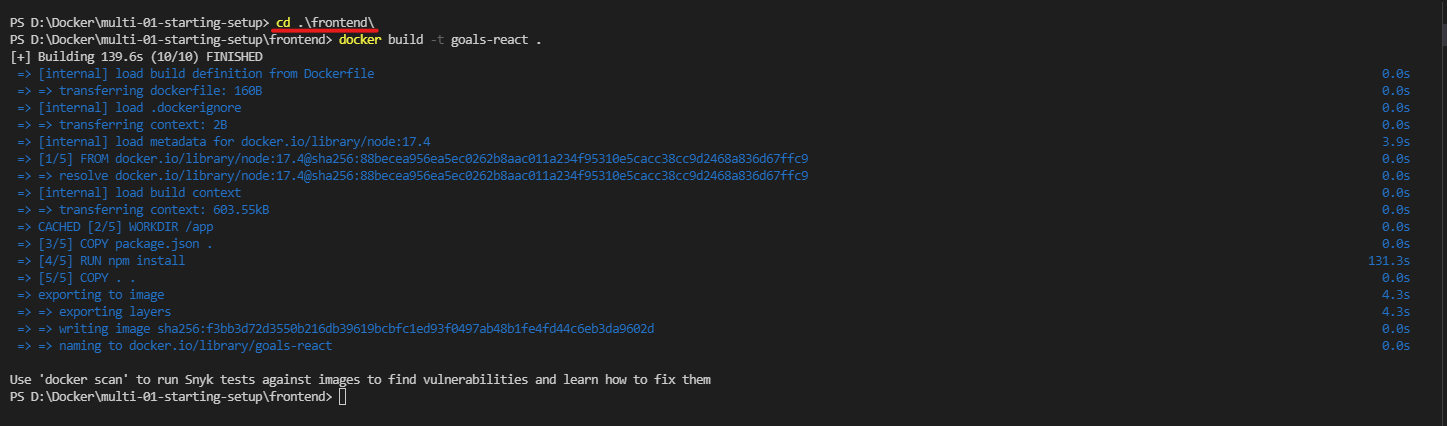


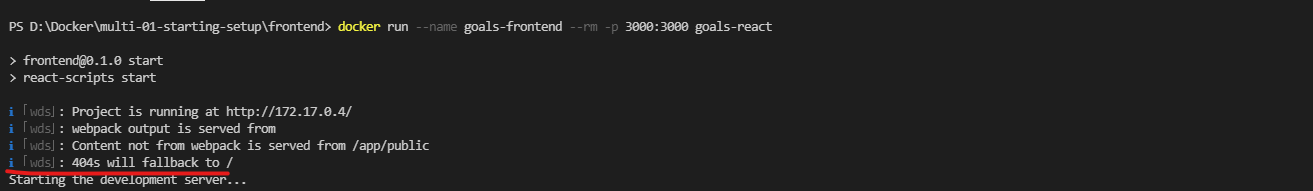


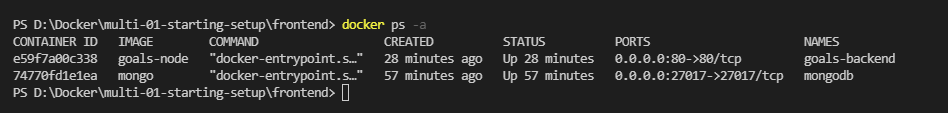
## Dockerizing React container

We need to write the customized Dockerfile for this frontend server also. This frontend server also depends Node Application, it’s not fully nodeJS code, but it will depends on it. So we need to custom NodeJS image by writing our instructions to it.

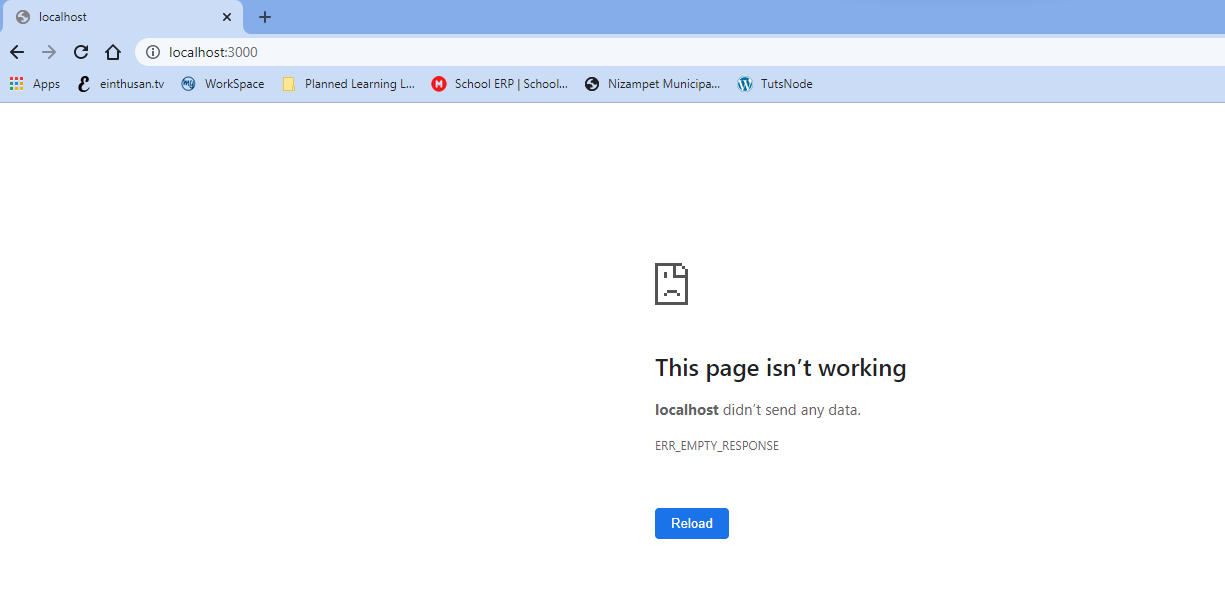








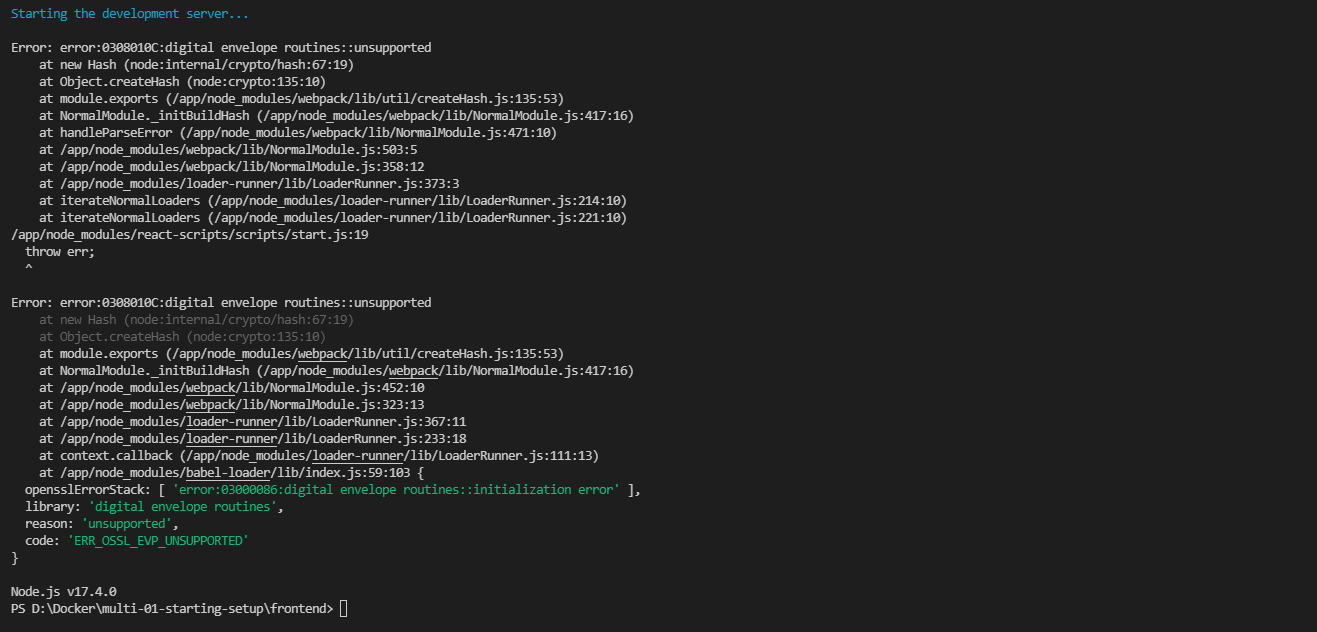
But when I browse this application, it’s not running. Something seems to be wrong here.



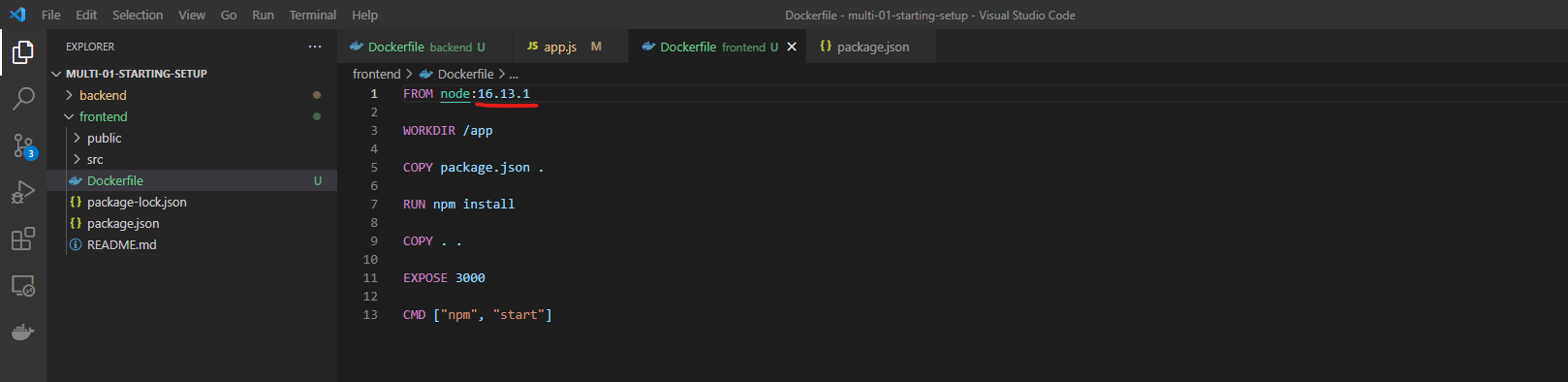
It’s because we had designed the REACT application to run interactive mode, which means always someone need to enter it. So we need to run the container in interactive mode by using the option -it. The react project is setup if it doesn’t receive any input it will automatically shutdown.

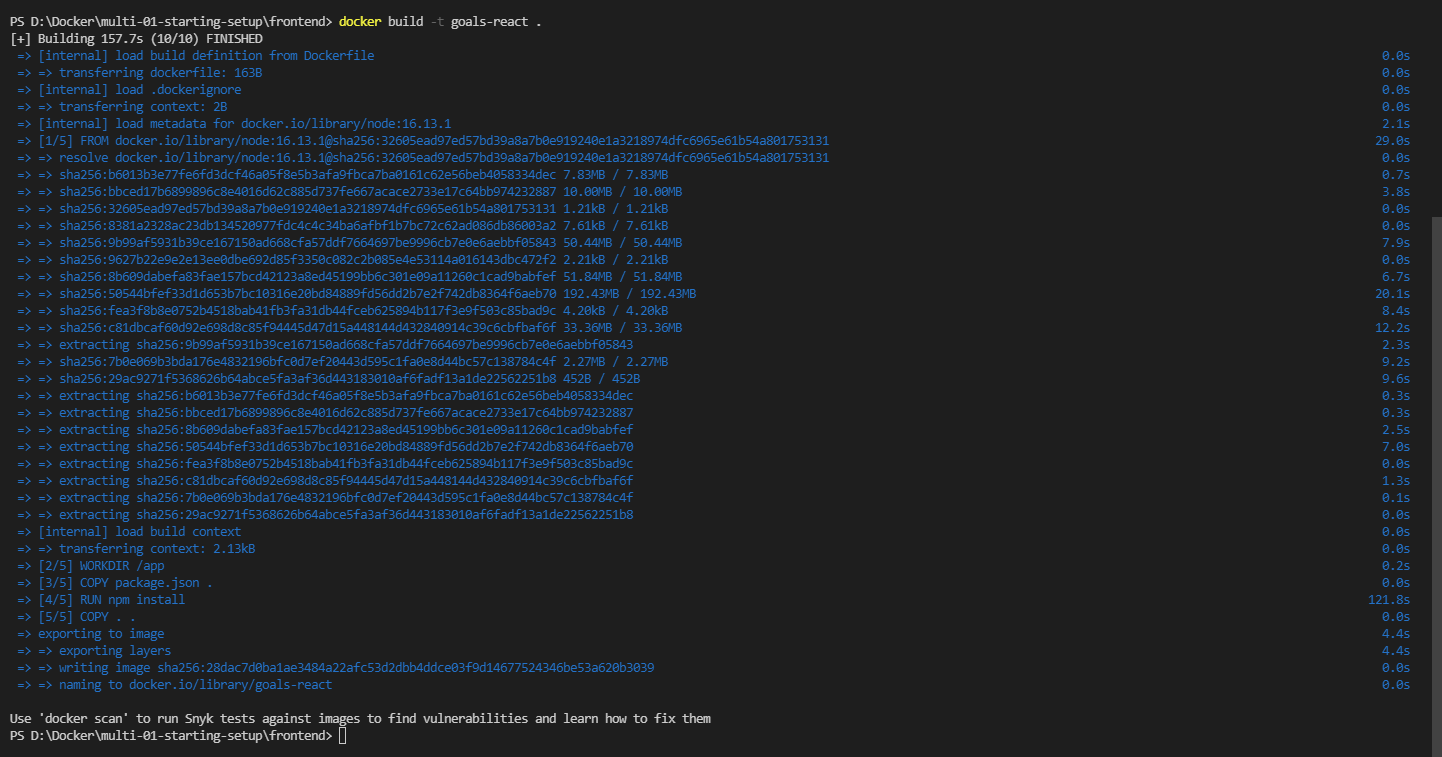
When I tried to start with the option with –it I got the below error.





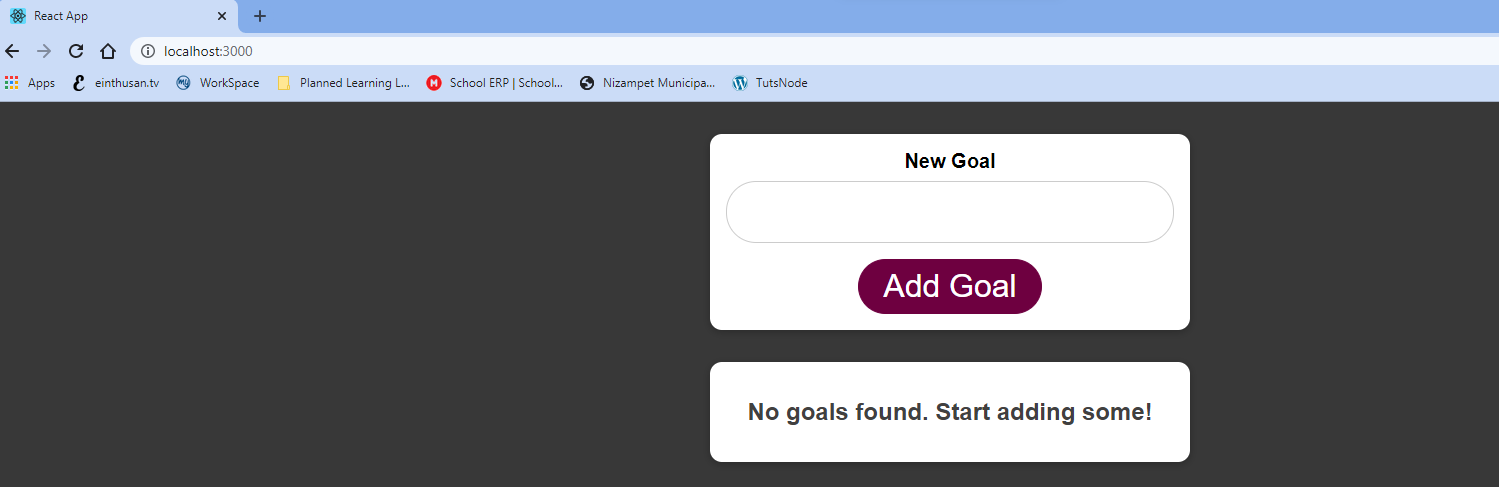
This bug is occurring on Node 17.4 version, so I am lowering the version to 16.13.1 as shown below.

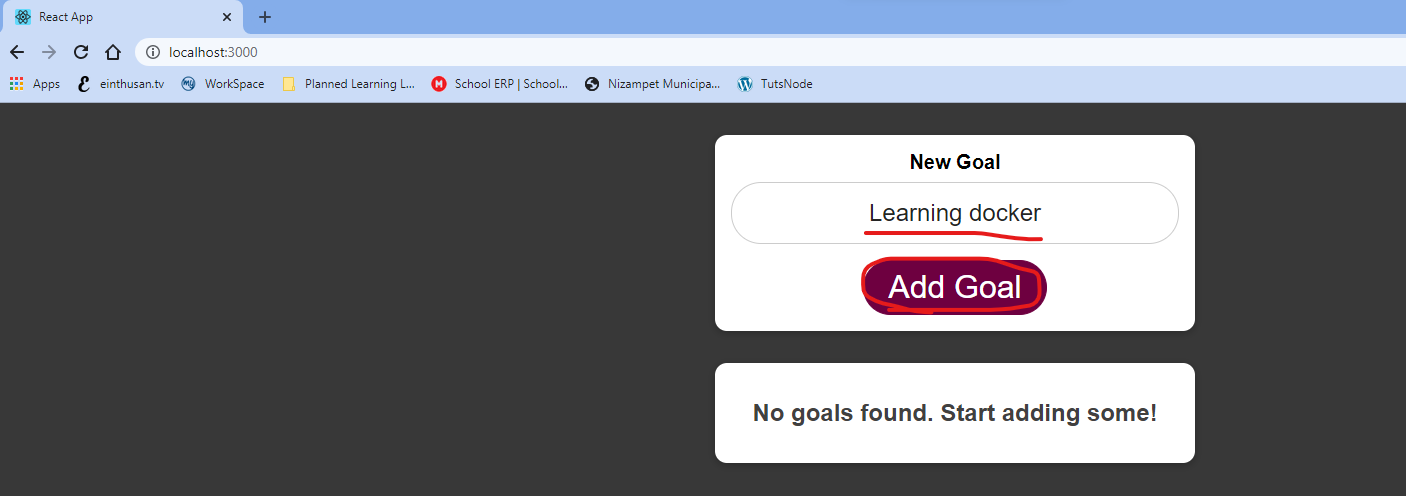


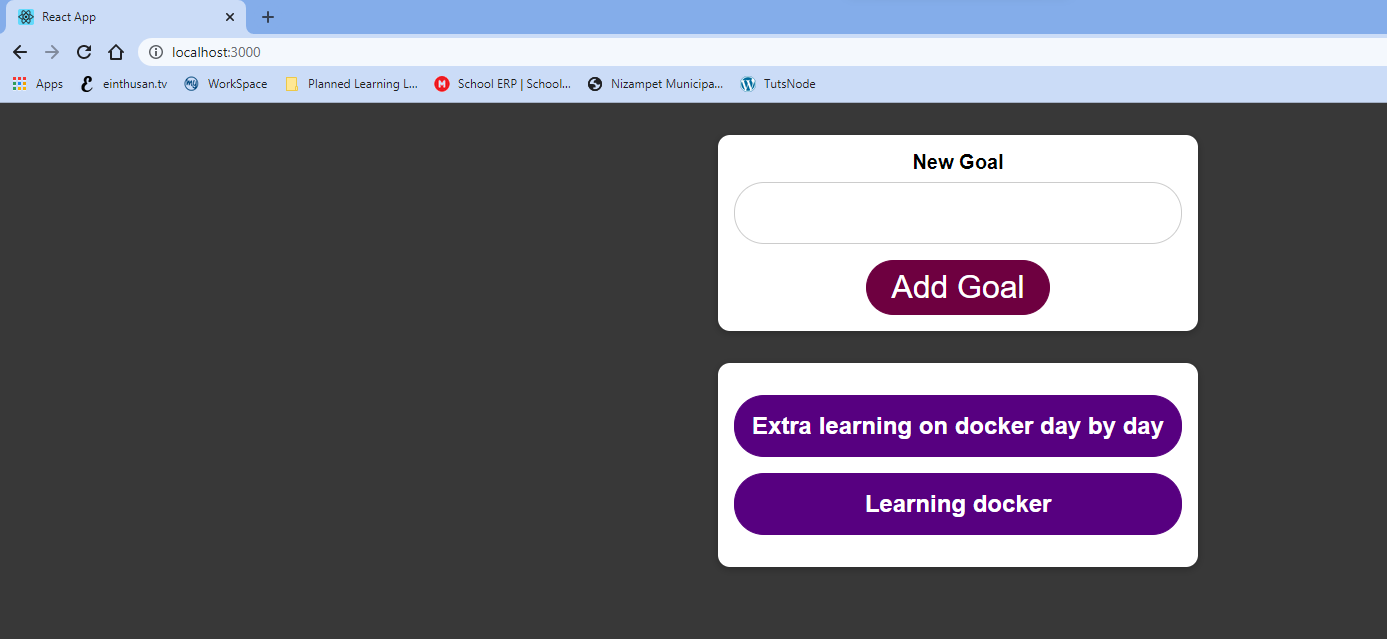








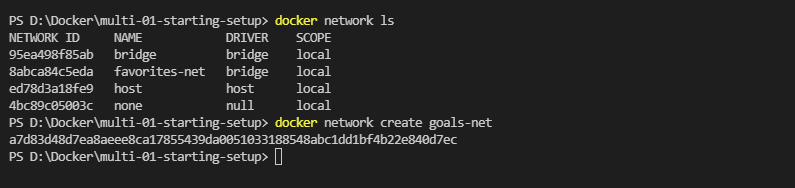


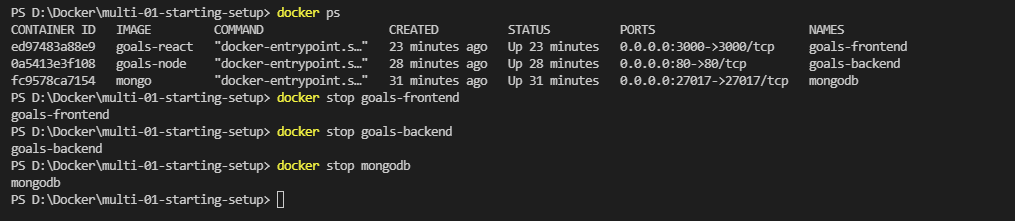


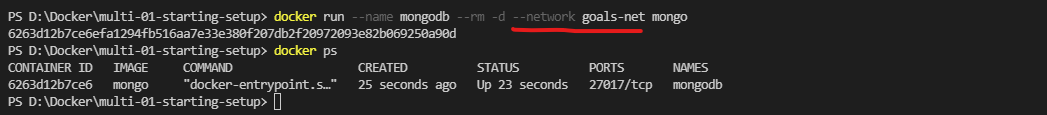
So as of now all of the containers are connecting to your locahost and running it. But as we have another efficient way by re-starting all of the containers in a single docker will automatically communicate, then we can remove the special domain “host.docker.internal” and replace with the container names in order to connect.

Now let’s make all the containers to start and communicate on a single network.

Let’s create the docker network first with the following command

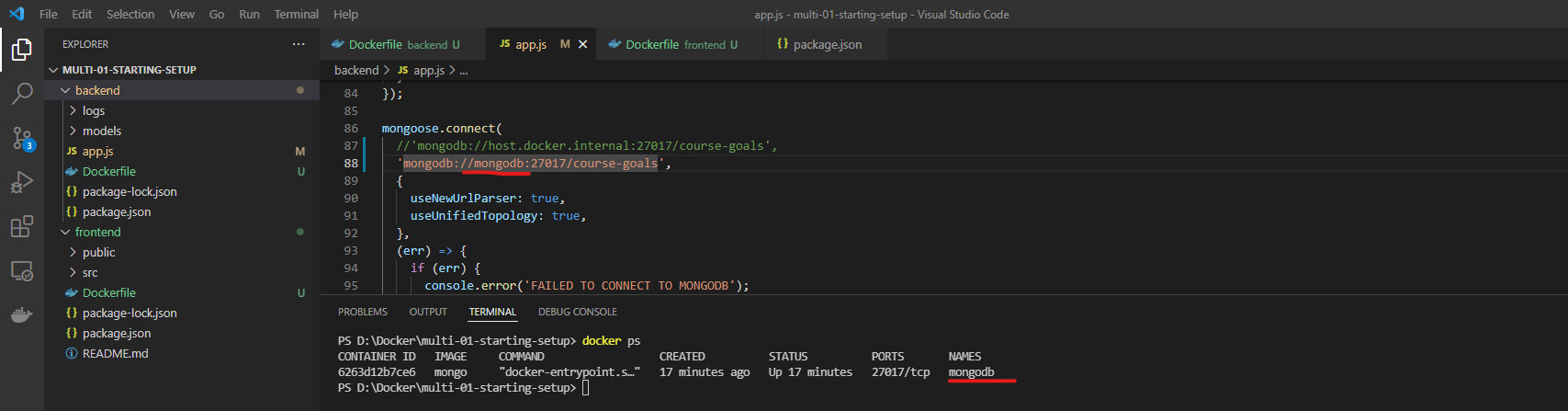


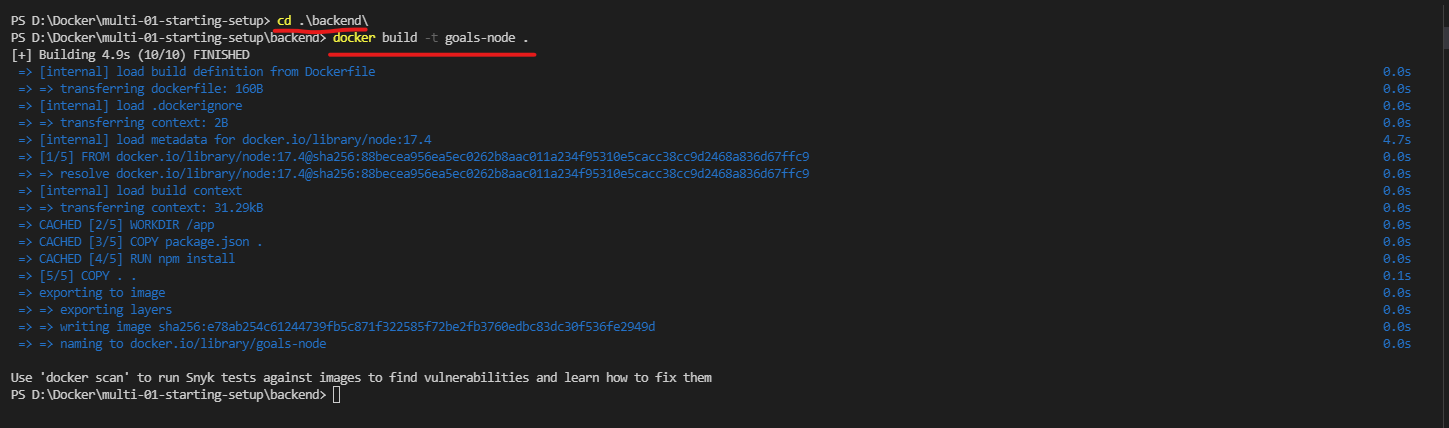


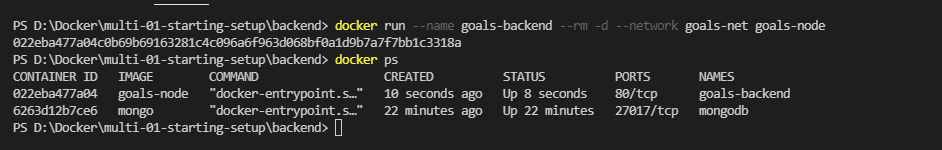


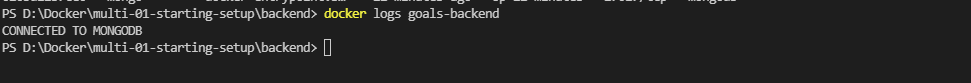
Let’s start the backend container

As we had started the mongodb container without publishing the port, so we need to place the mongodb container name on the mongodb url.



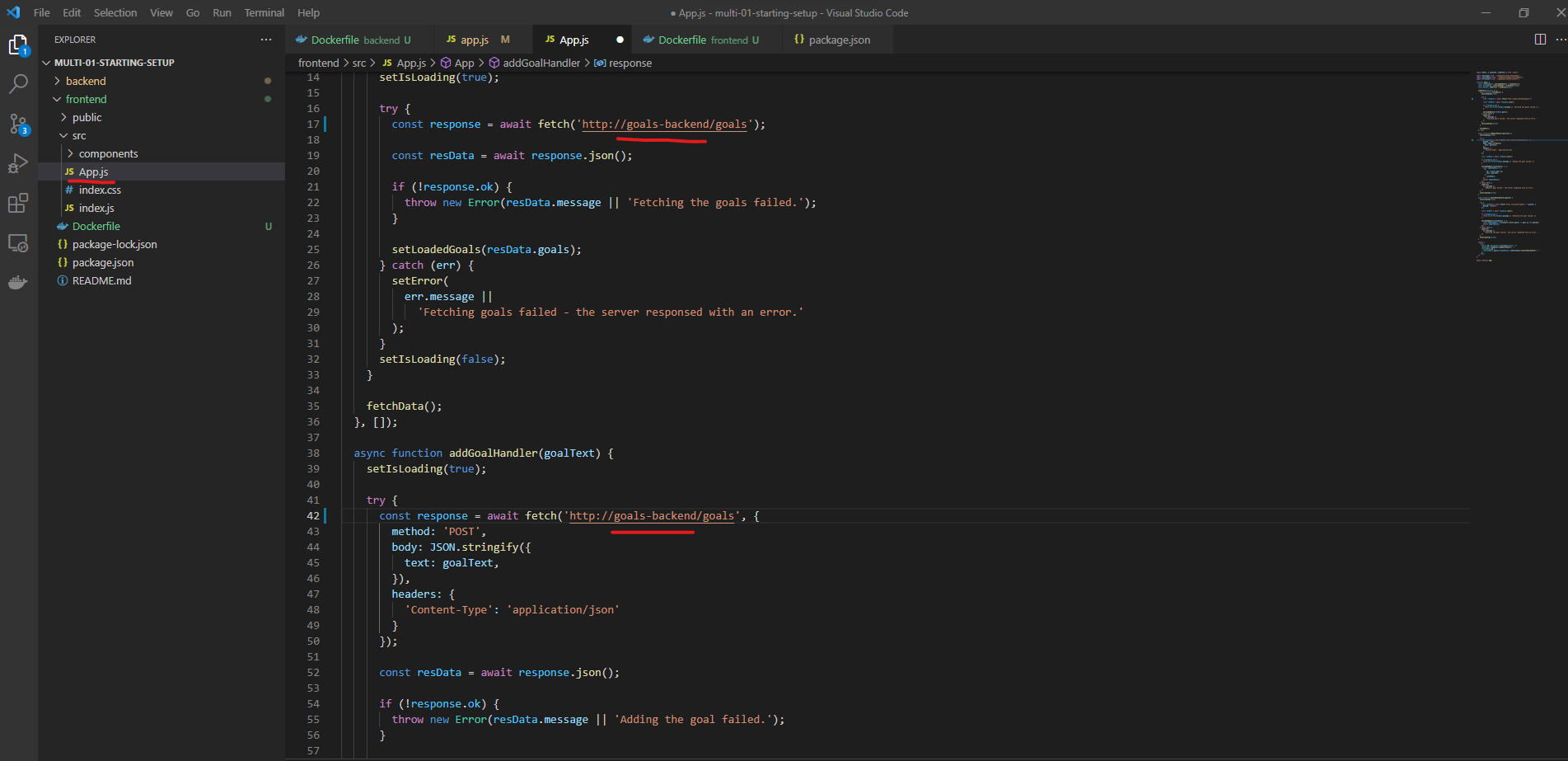


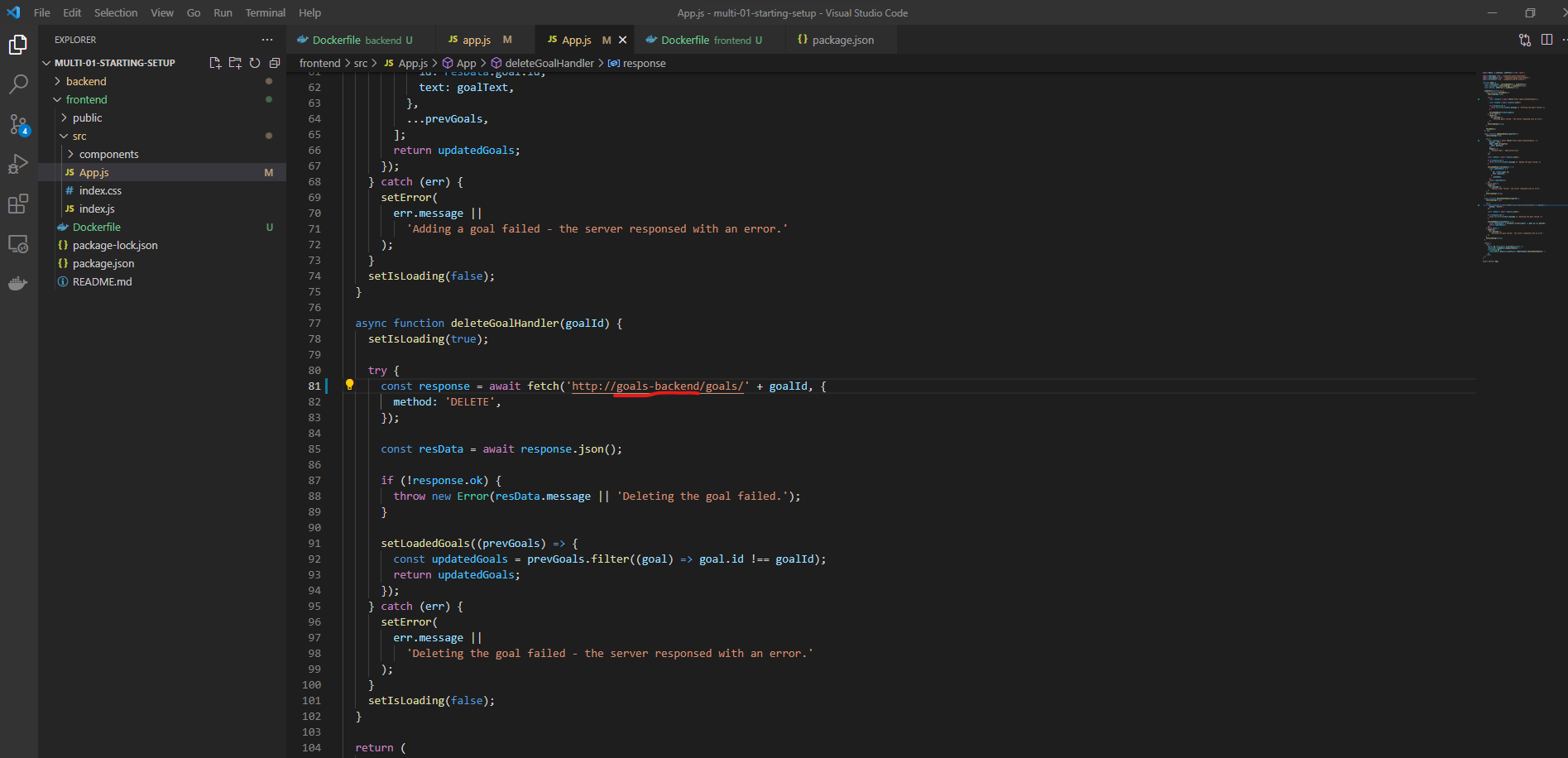


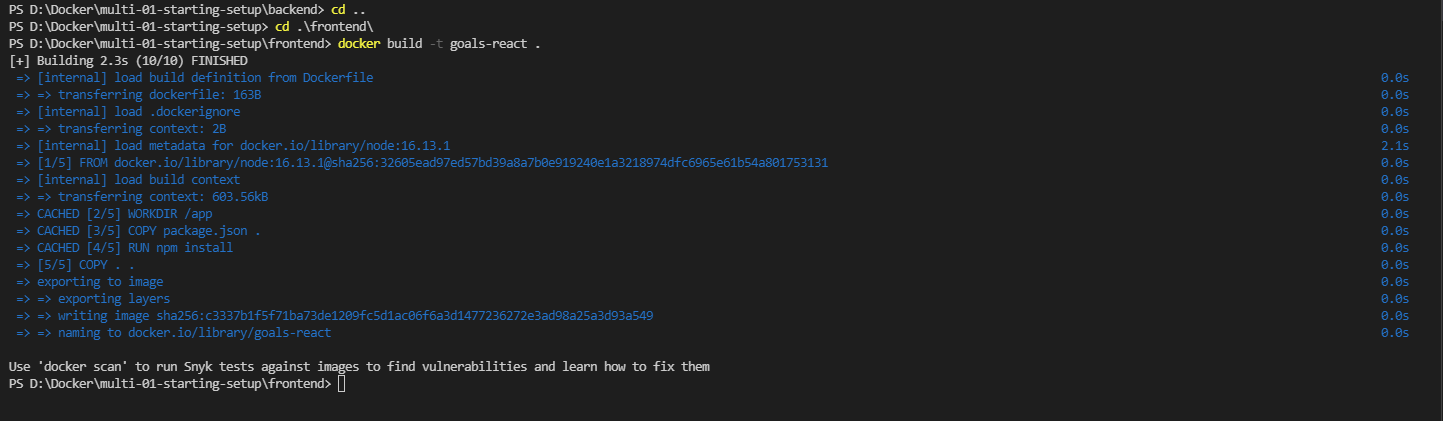


Now let’s start the front end React single page application (SPA)

Before starting this container, we need to modify the app.js file by replacing the localhost with the backend container name.



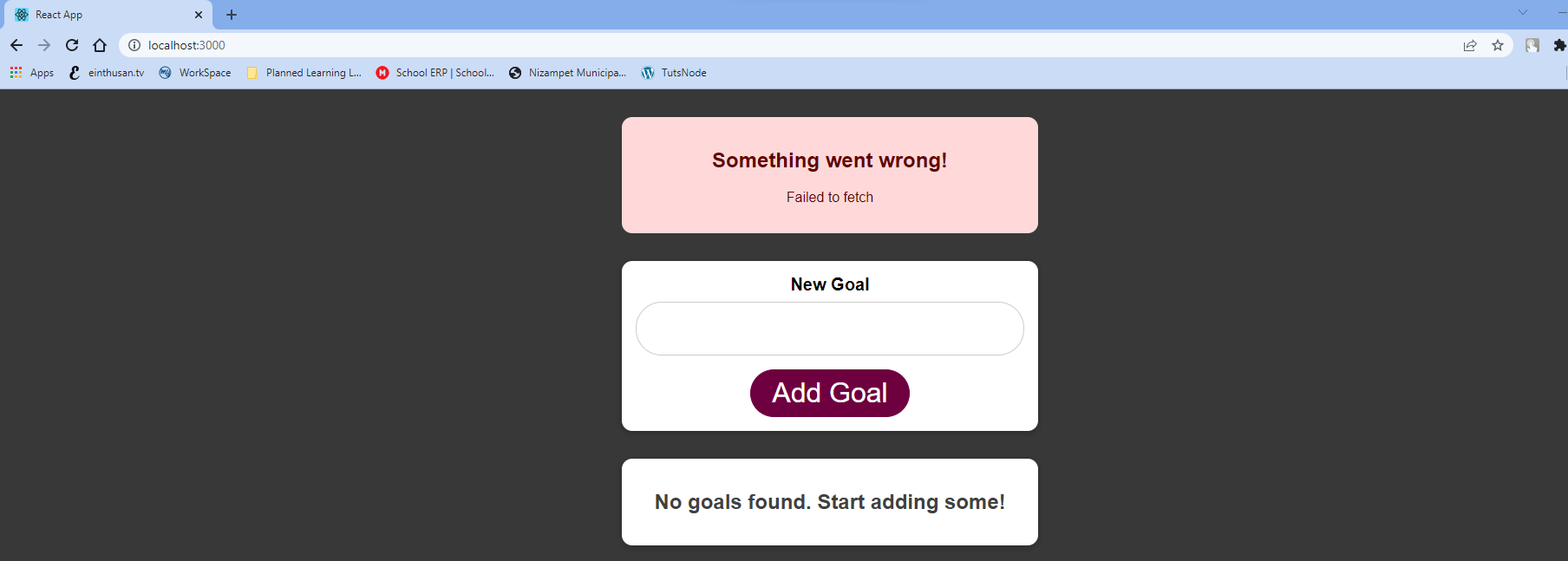




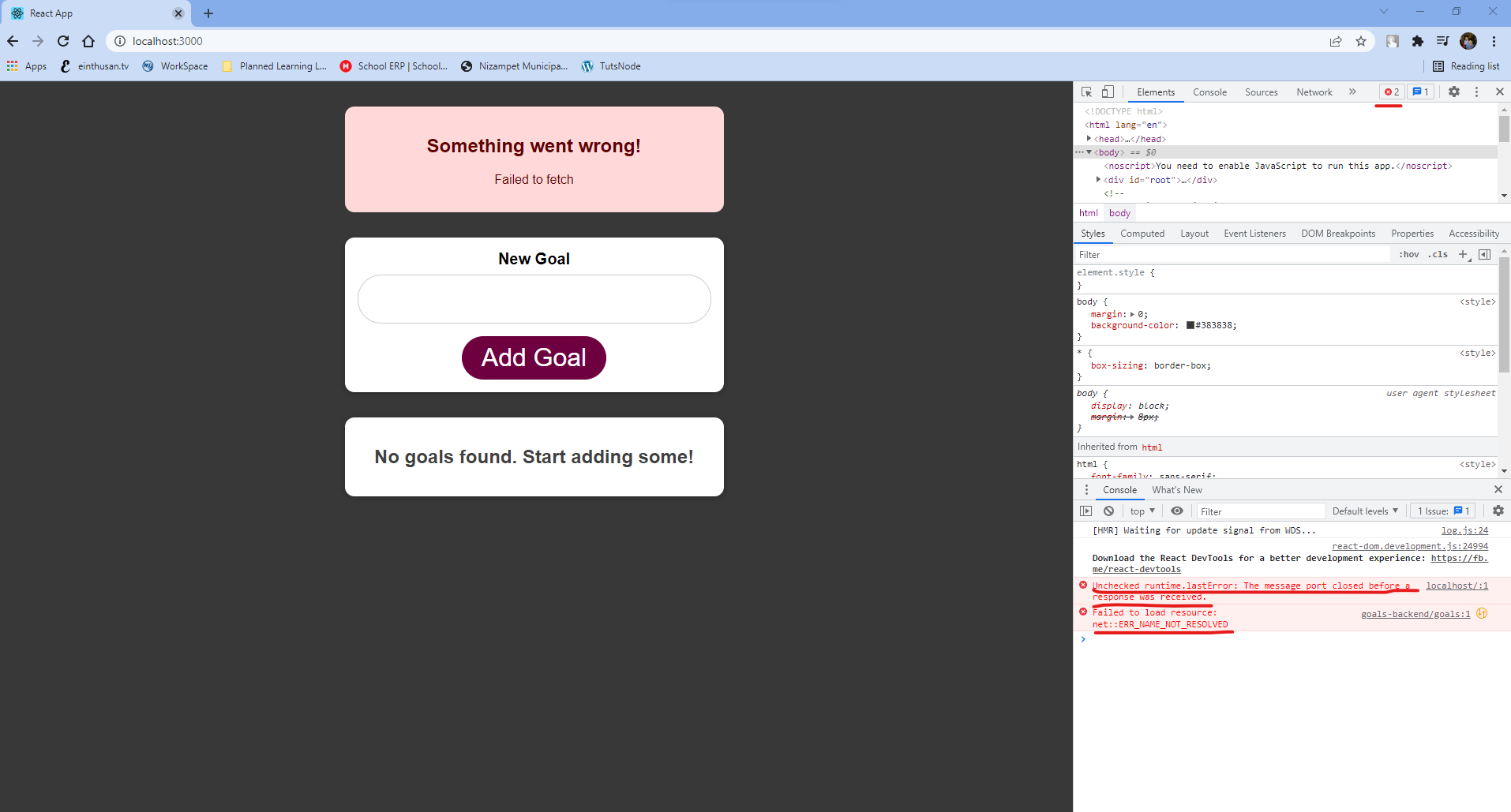


Note: Here I am still publish the port as I want this react to test it from my localhost on browser.



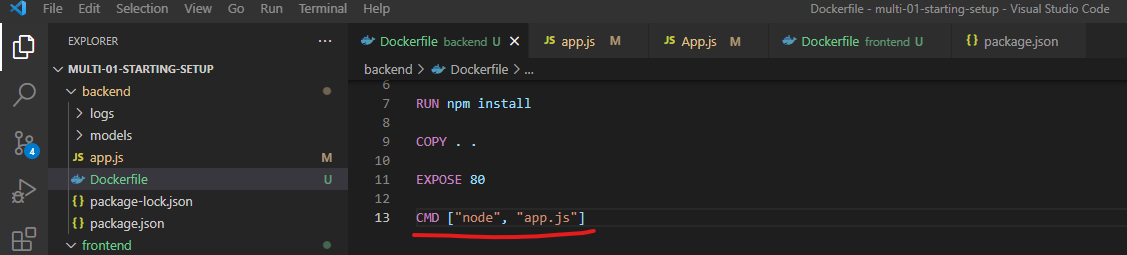


Here I got an error, in order to see the Hit F12 it will open a source code click on cross icon to view the error.

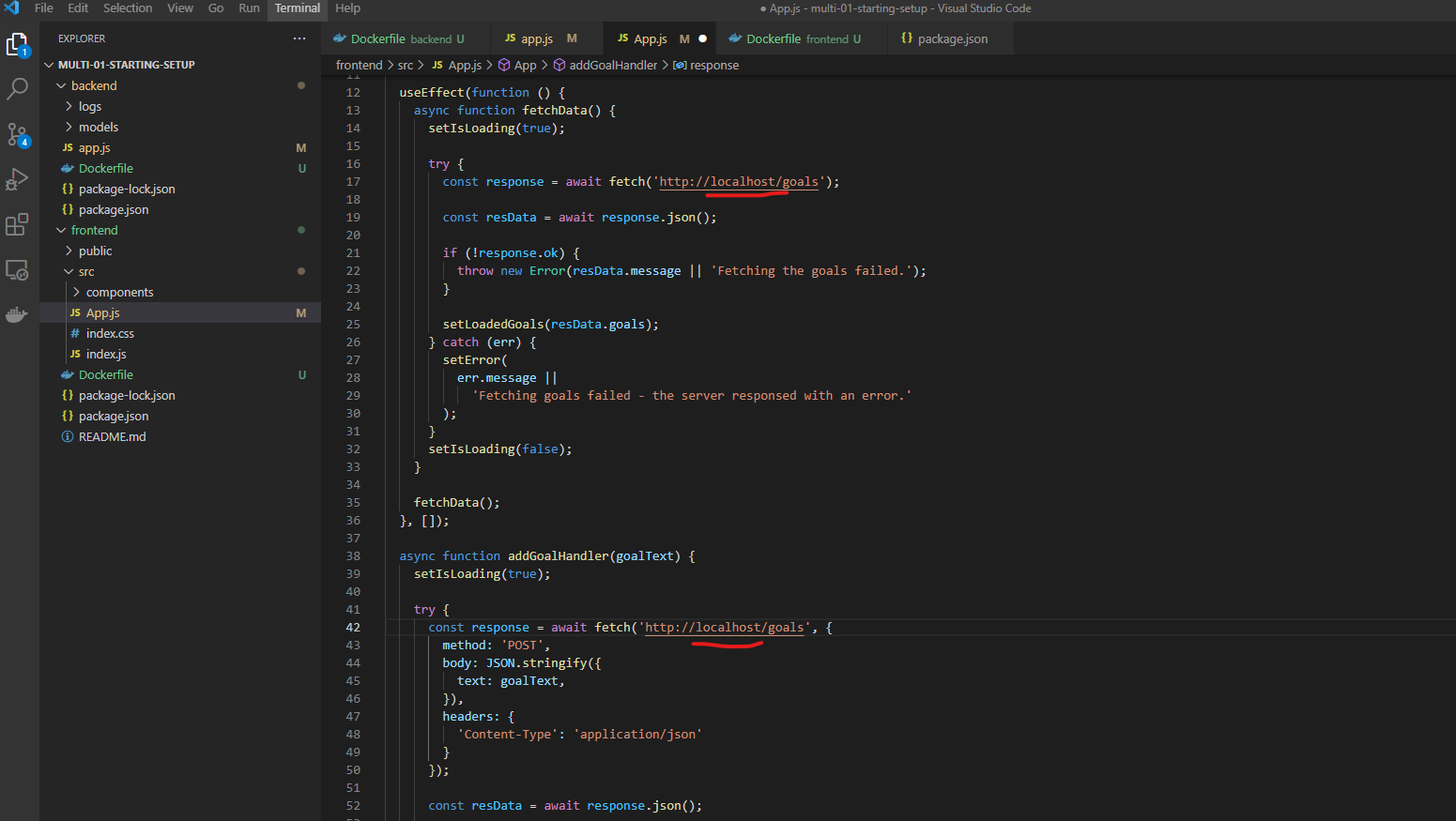


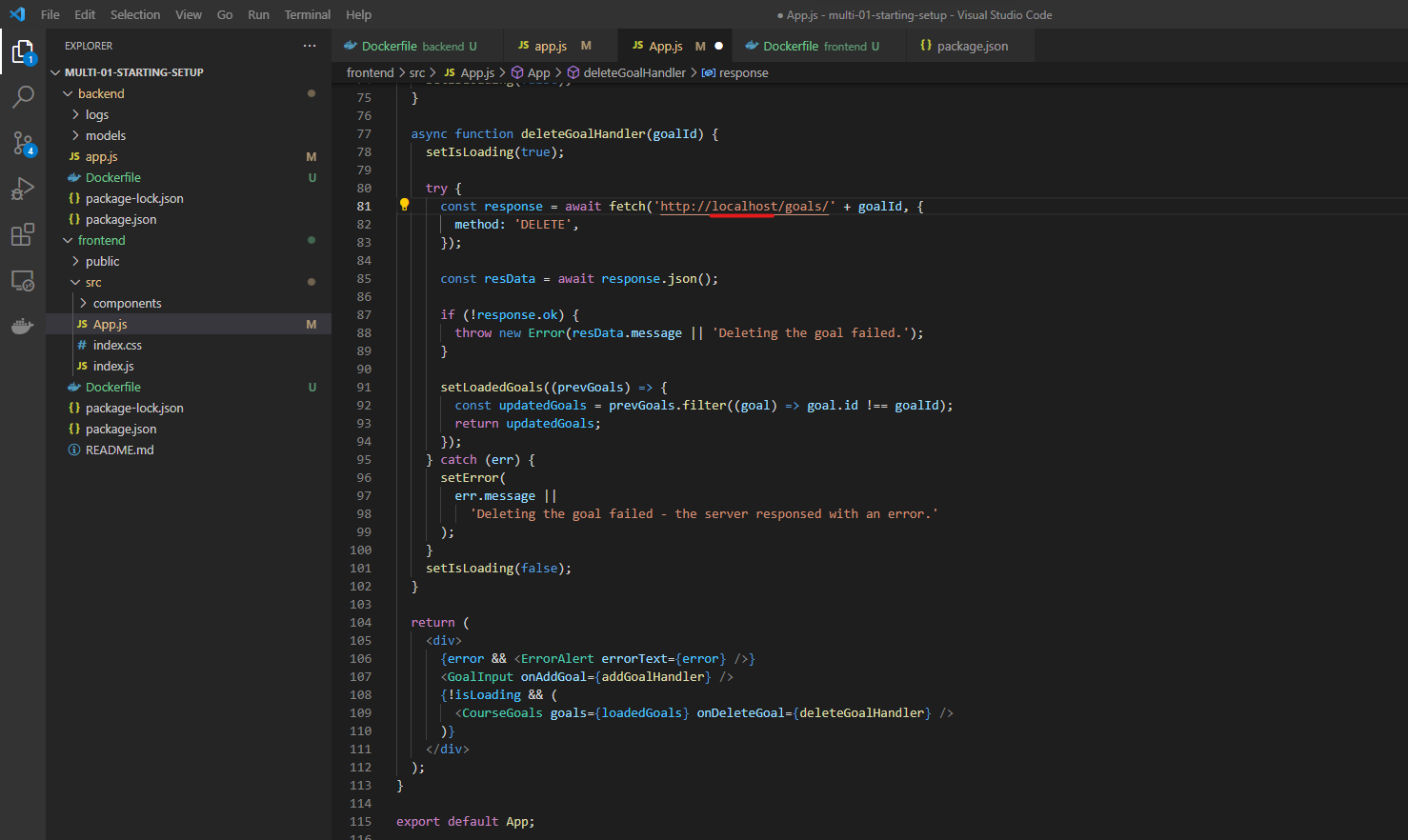
In order to understand the REACT, normally the from end java script code will run on browser not on container that’s the key difference compared to the backend server node code.

On back end server, it runs the following command on server

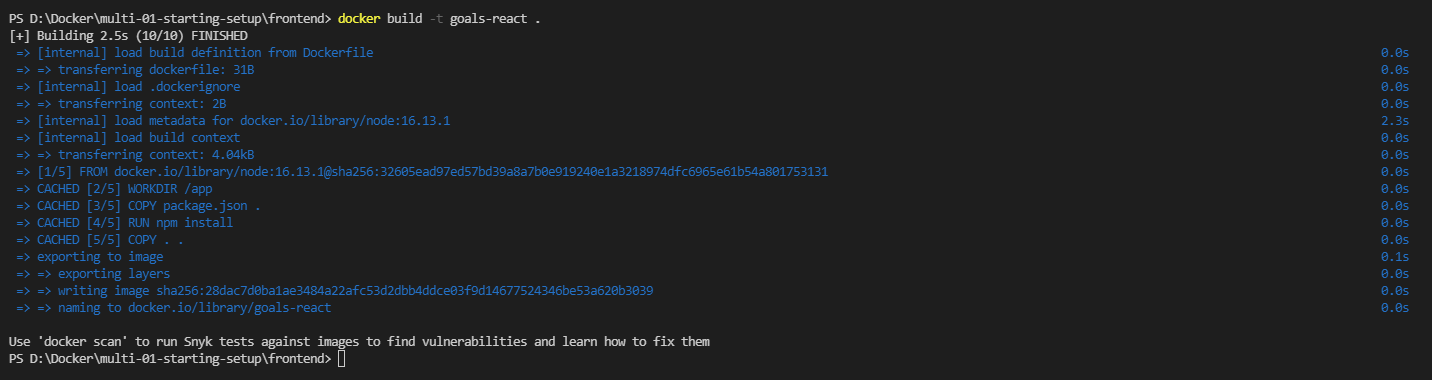


But on front end server, the command “npm start” it will does only one thing it will start basic development server which serves basic REACT code. This react code will not run on server, it will run on browser on localhost. So previously which ever we point the goals-backend which will run the code on server, so the code run localhost browser. So we need to revert back to localhost there, the only thing run on front end container is the development server, not the nodejs code.





Which mean we still need to publish the ports as the code executes on localhost, Now we need to rebuild the code again.



While running the front end container we don’t need --network as the code is not running container it is running on localhost.





Now we need to restart the backend server to publish the port as it front server is communicating with it. Also it needs a network because our mongodb is communicating this backend server.

