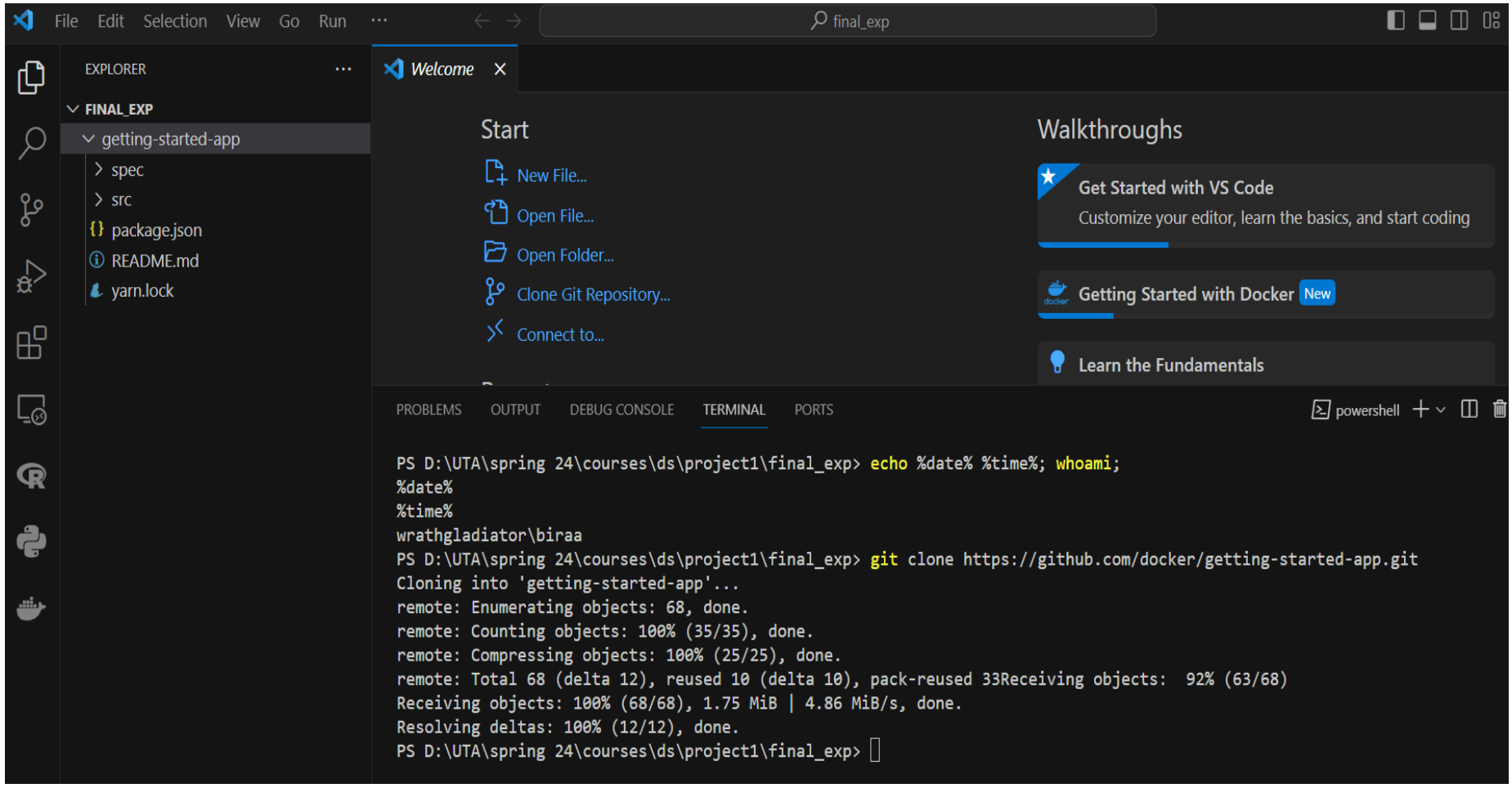
NAME: BIRAAJ ROUT UTA ID: 1002071886

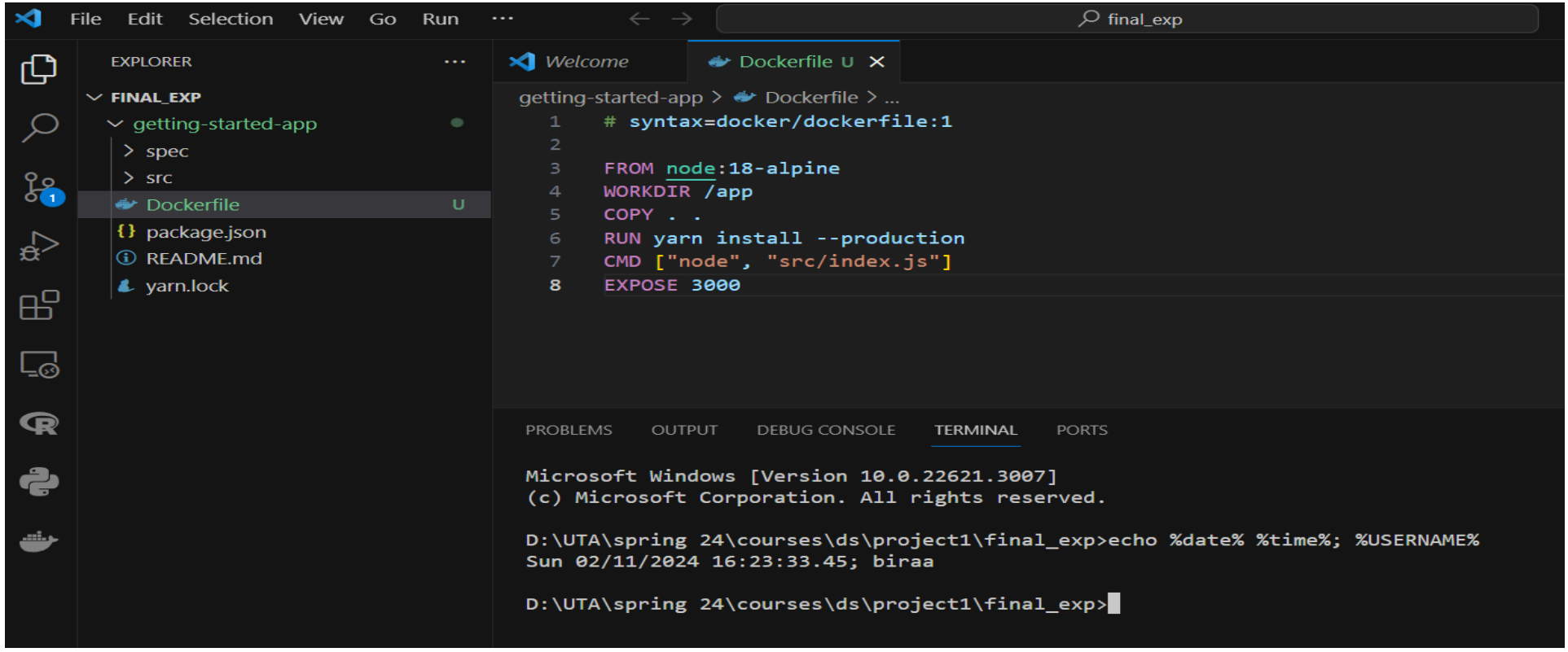
Project Assignment 1: Getting Started with Docker Report

1. Get started guide:
    a. Important link
        i. **Link to view the docker image**: https://hub.docker.com/repository/docker/biraaj/getting-started
    b. Part 1: Overview of the get started guide
        i. **Container**: sandboxed process running on a host machine that is isolated from all other processes running on that host machine.
        ii. **Image**: An image is a read-only template with instructions for creating a Docker container
    c. Part 2: how to build and run an image as a container
        i. Clone the getting-started-app repository
            1. Command: git clone https://github.com/docker/getting-started-app.git
            2. **Git clone:** The git clone command is used to create a copy of a specific repository or branch within a repository.



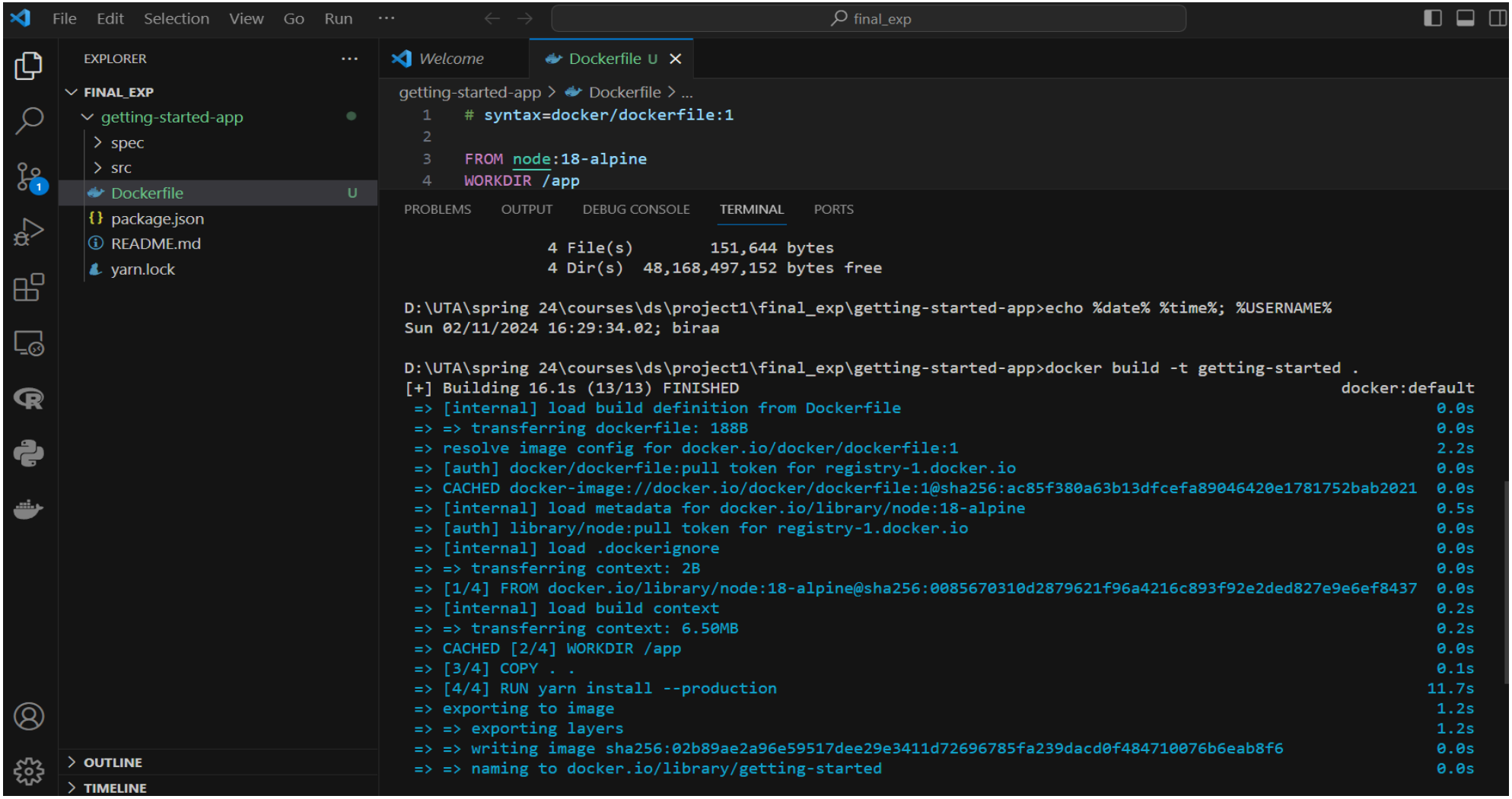
ii. Creating Docker File

1. **Dockerfile:** A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.



iii. Build the image

1. Command: **docker build -t getting-started .**
    a. The docker build command builds Docker images from a Dockerfile.
    b. -t option here is to Name the docker build

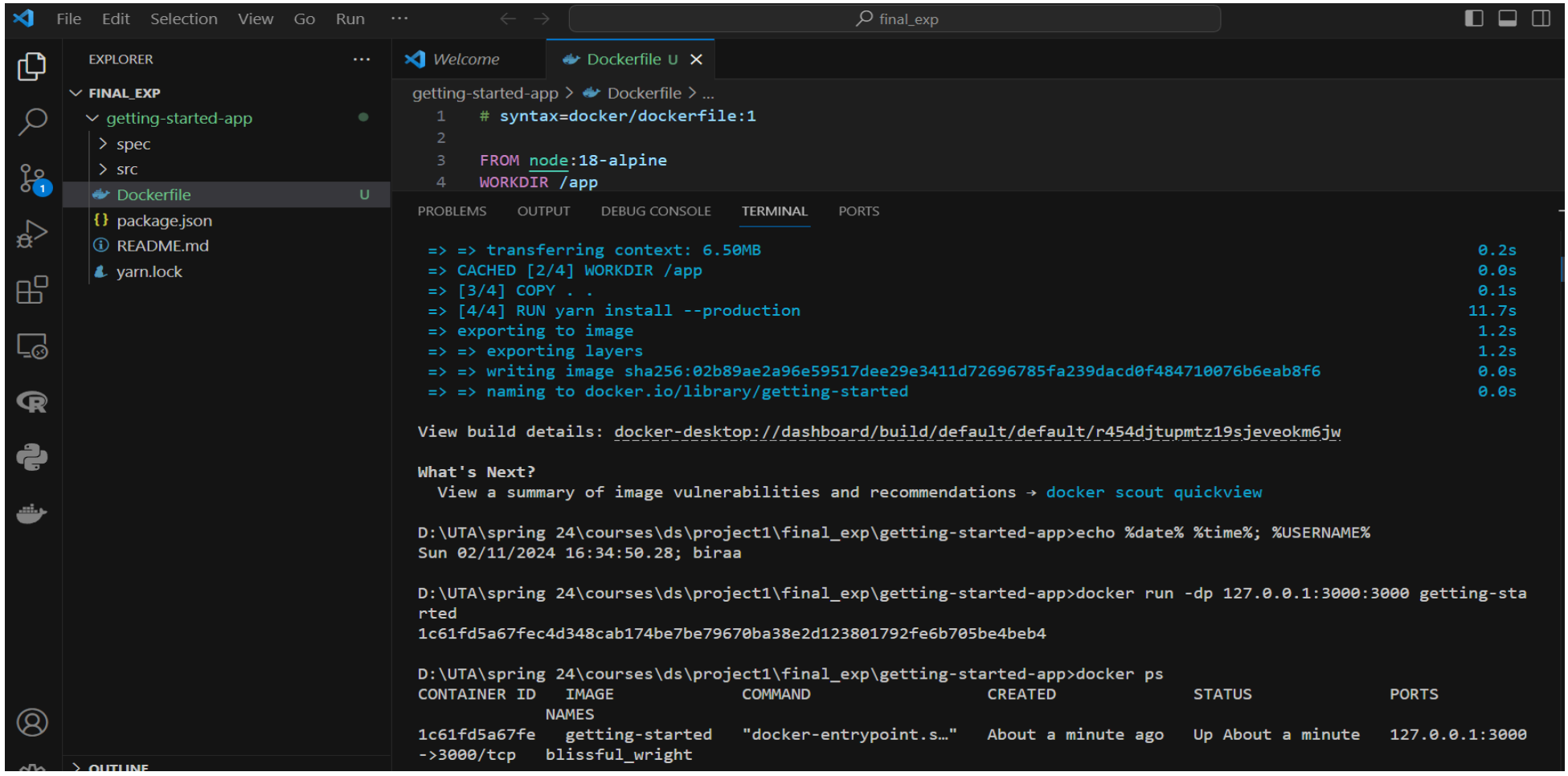


iv. Start an app container

1. Command: **docker run -dp 127.0.0.1:3000:3000 getting-started**
    a. Docker run command is used to run a docker container.

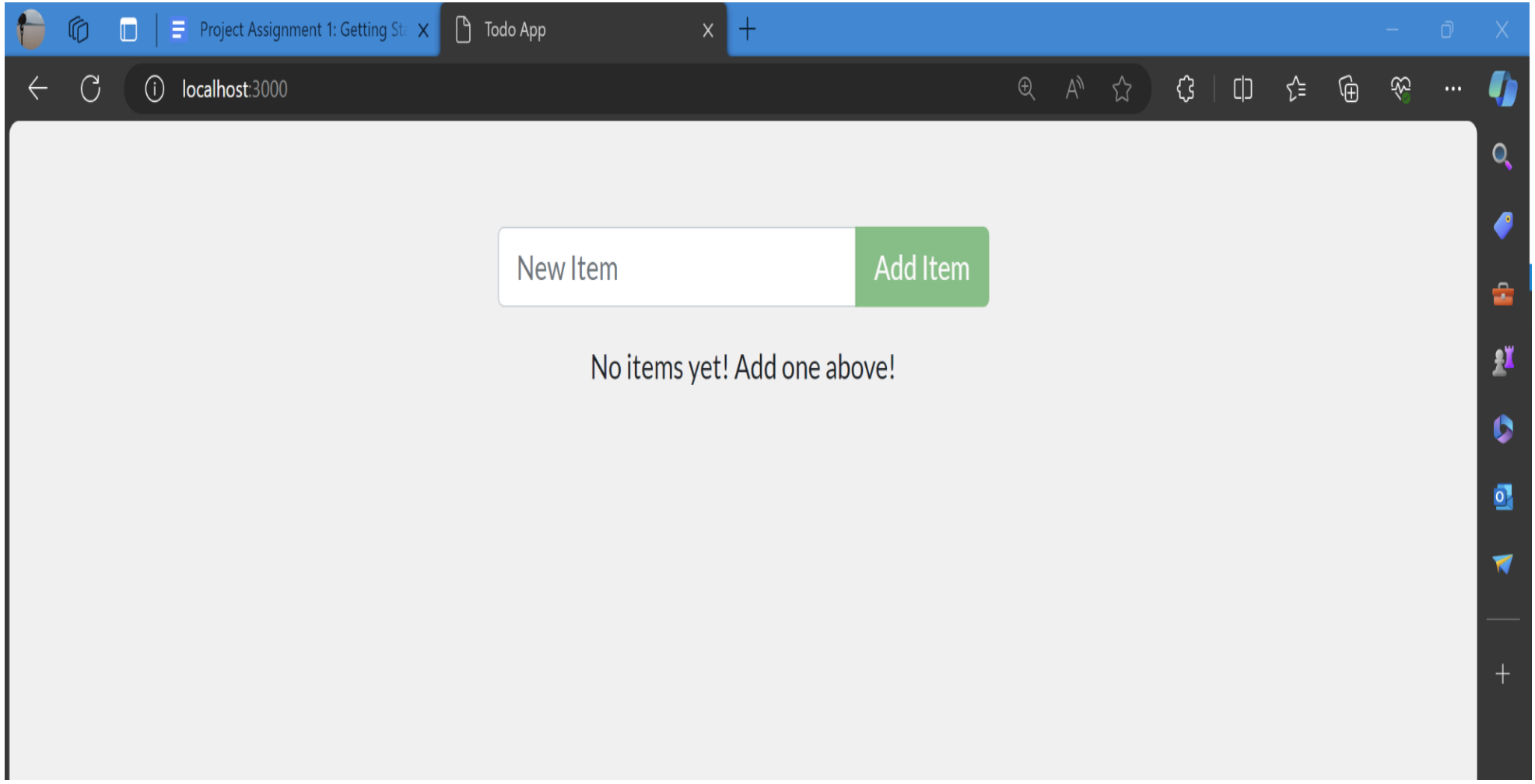
b. Here -d runs the container in background
c. -p stands for –publish and creates a port mapping between host and a container. I.e 127.0.0.1:3000:3000 and here the container is **getting-started**

2. Command: **docker ps**
   a. This command lists all the running docker processes or containers.



v. Viewing the app after the container runs

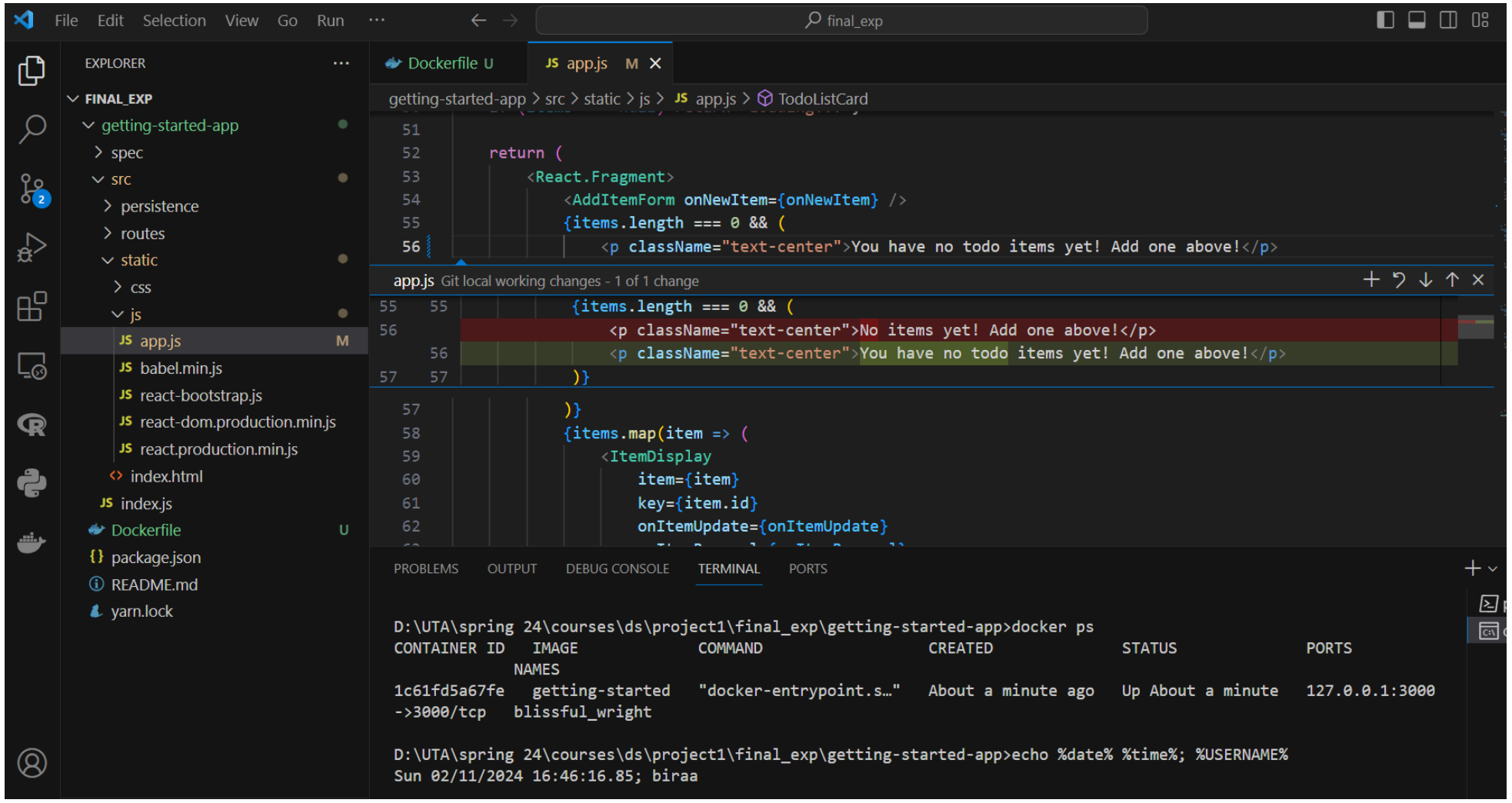
1. Url: http://localhost:3000



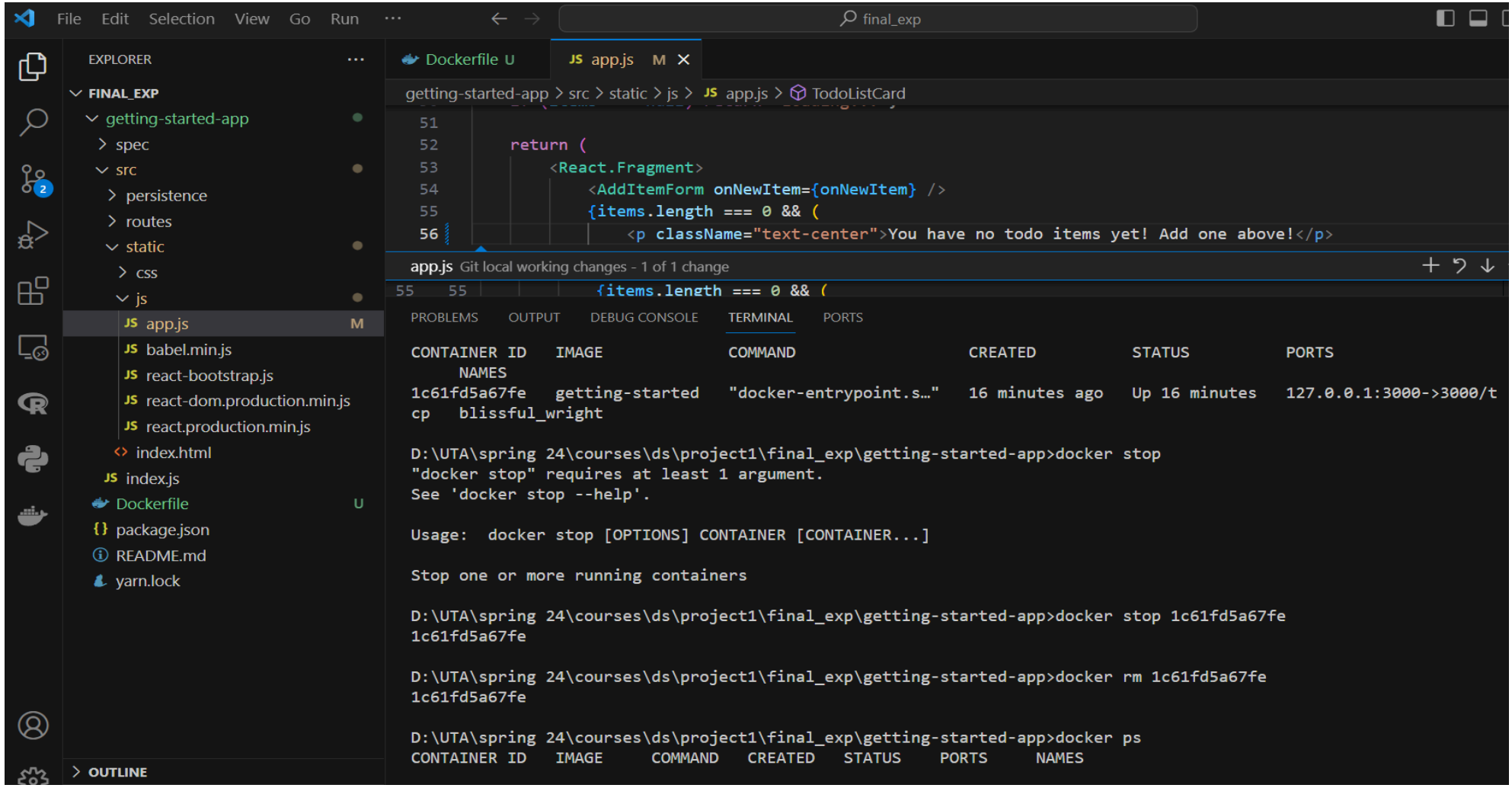
d. Part 3: Update the application

i. Update the source code

1. Here we change some lines in app.js to see the updates being reflected after the build.

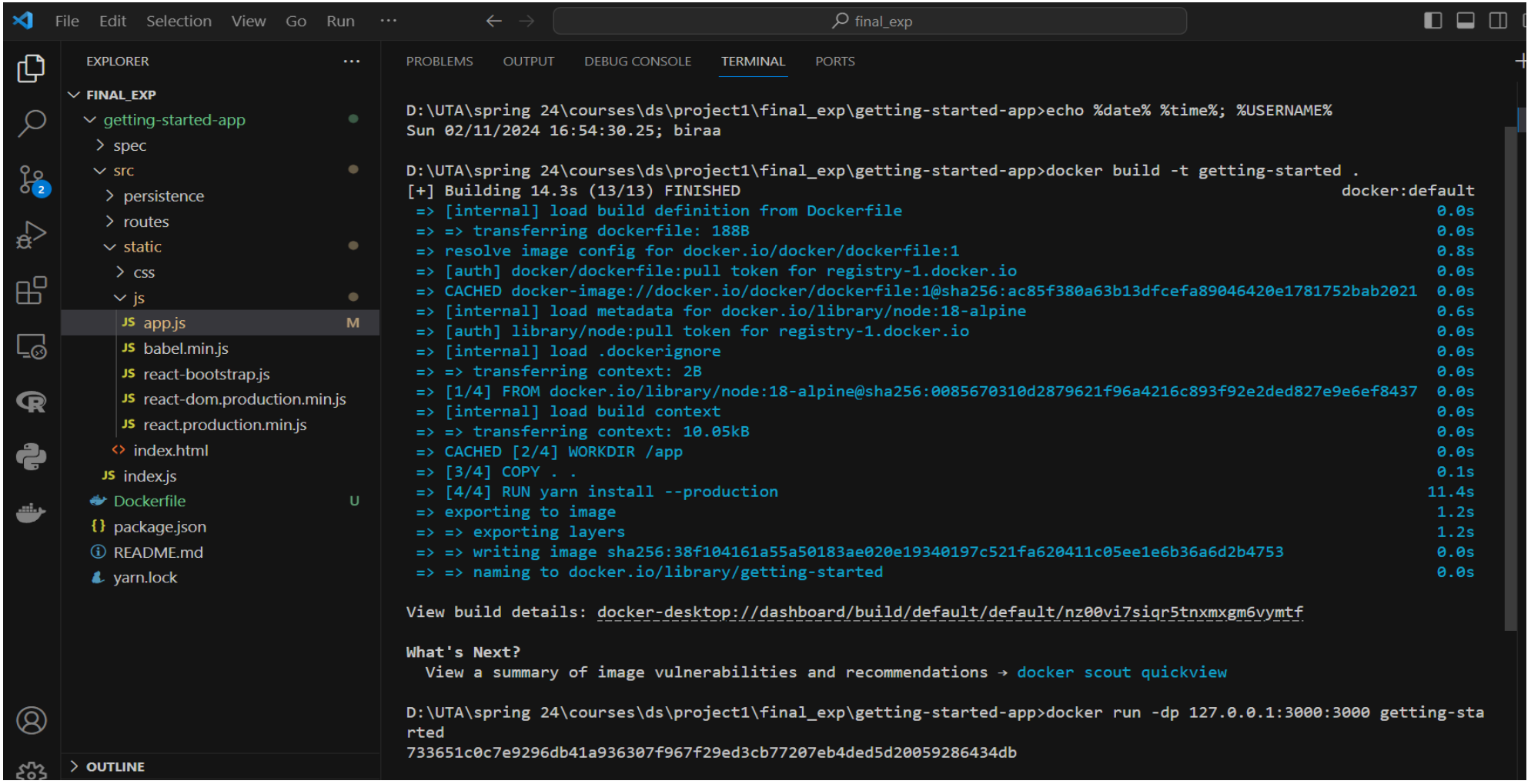


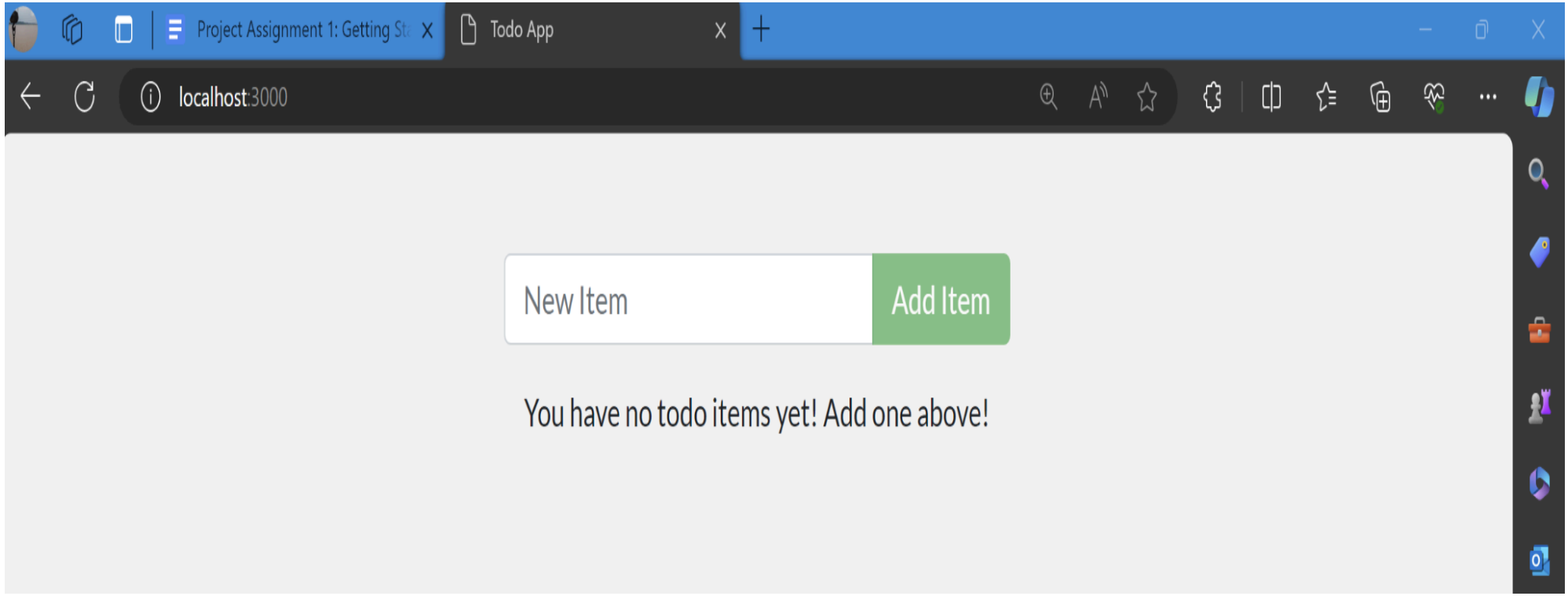
ii. Remove the old container using CLI:
    1. Commands:
        a. **docker ps** (explained in part 2)
        b. **docker stop 1c61fd5a67fe**
            i. This command stops the run container. Here the container ID is **1c61fd5a67fe**
        c. **docker rm 1c61fd5a67fe**
            i. This command removes the container from the process list. Here we remove container **1c61fd5a67fe**



iii. Build and Run the updated container

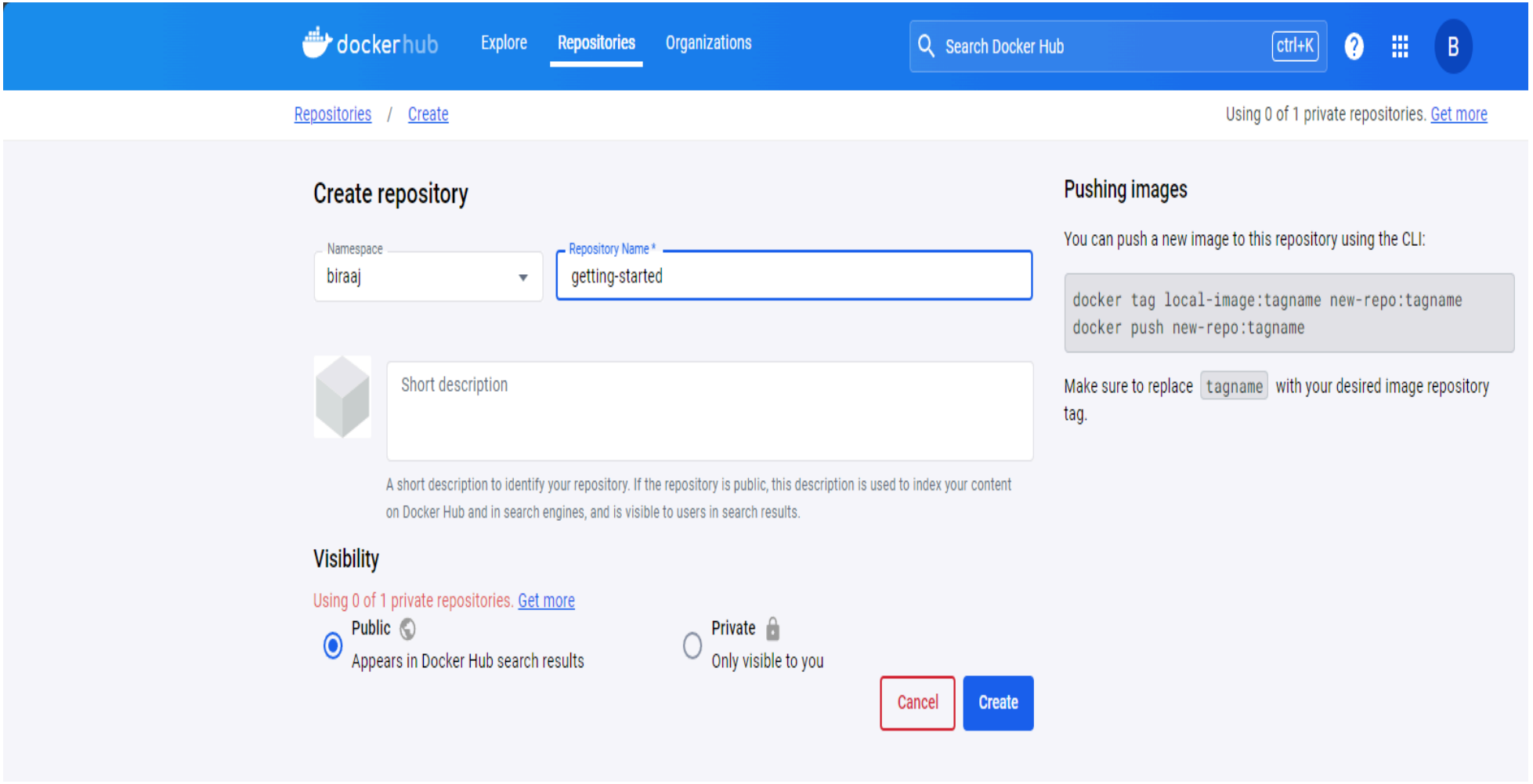
1. **docker build -t getting-started .**(Explained in part2)
2. **docker run -dp 127.0.0.1:3000:3000 getting-started**(Explained in part2)


File Edit Selection View Go Run ···
final_exp
EXPLORER
FINAL_EXP
getting-started-app
spec
src
persistence
routes
static
css
js
app.js M
babel.min.js
react-bootstrap.js
react-dom.production.min.js
react.production.min.js
index.html
index.js
Dockerfile U
package.json
README.md
yarn.lock
OUTLINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
D:\UTA\spring 24\courses\ds\project1\final_exp\getting-started-app>echo %date% %time%; %USERNAME%
Sun 02/11/2024 16:54:30.25; biraa
D:\UTA\spring 24\courses\ds\project1\final_exp\getting-started-app>docker build -t getting-started .
[+] Building 14.3s (13/13) FINISHED docker:default
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 188B 0.0s
=> resolve image config for docker.io/docker/dockerfile:1 0.8s
=> [auth] docker/dockerfile:pull token for registry-1.docker.io 0.0s
=> CACHED docker-image://docker.io/docker/dockerfile:1@sha256:ac85f380a63b13dfcefa89046420e1781752bab2021 0.0s
=> [internal] load metadata for docker.io/library/node:18-alpine 0.6s
=> [auth] library/node:pull token for registry-1.docker.io 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [1/4] FROM docker.io/library/node:18-alpine@sha256:0085670310d2879621f96a4216c893f92e2ded827e9e6ef8437 0.0s
=> [internal] load build context 0.0s
=> => transferring context: 10.05kB 0.0s
=> CACHED [2/4] WORKDIR /app 0.0s
=> [3/4] COPY . . 0.1s
=> [4/4] RUN yarn install --production 11.4s
=> exporting to image 1.2s
=> => exporting layers 1.2s
=> => writing image sha256:38f104161a55a50183ae020e19340197c521fa620411c05ee1e6b36a6d2b4753 0.0s
=> => naming to docker.io/library/getting-started 0.0s
View build details: docker-desktop://dashboard/build/default/default/nz00vi7siqr5tnxmxgm6vymtf
What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview
D:\UTA\spring 24\courses\ds\project1\final_exp\getting-started-app>docker run -dp 127.0.0.1:3000:3000 getting-started
733651c0c7e9296db41a936307f967f29ed3cb77207eb4ded5d20059286434db



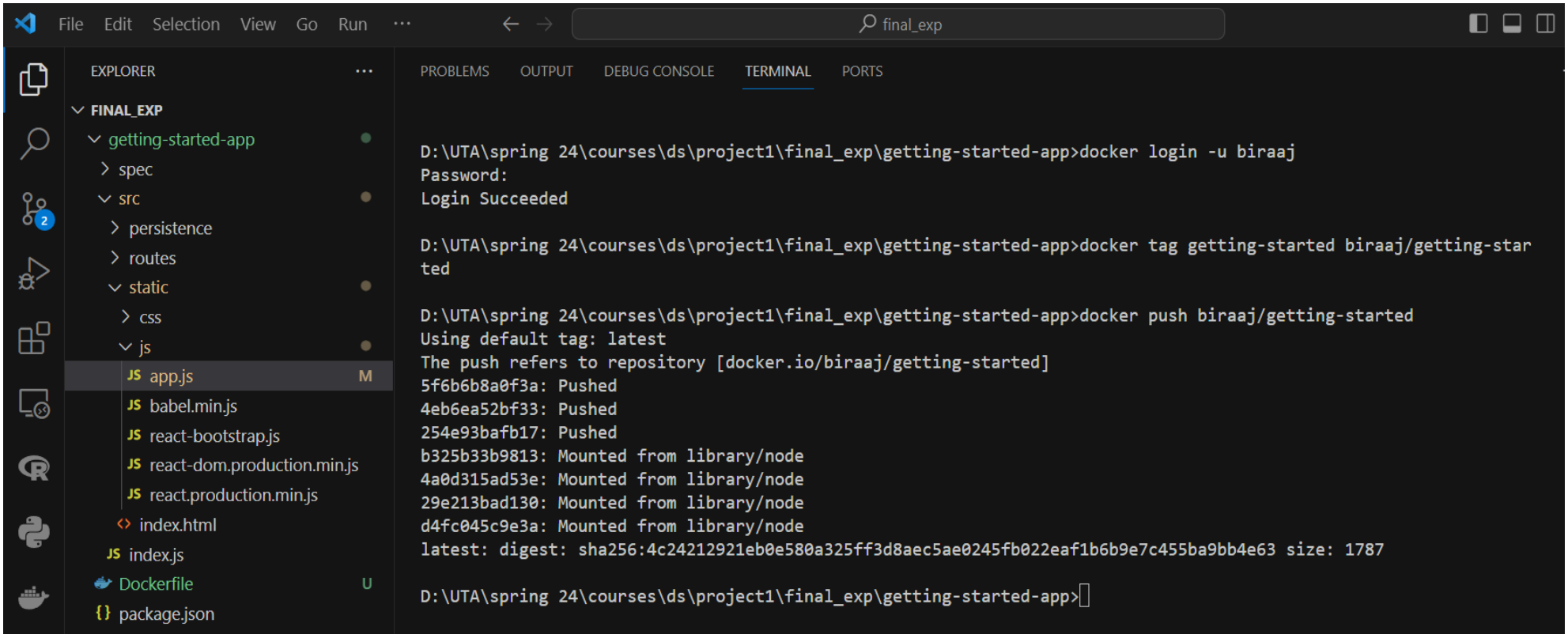
e. Part 4: Share the application

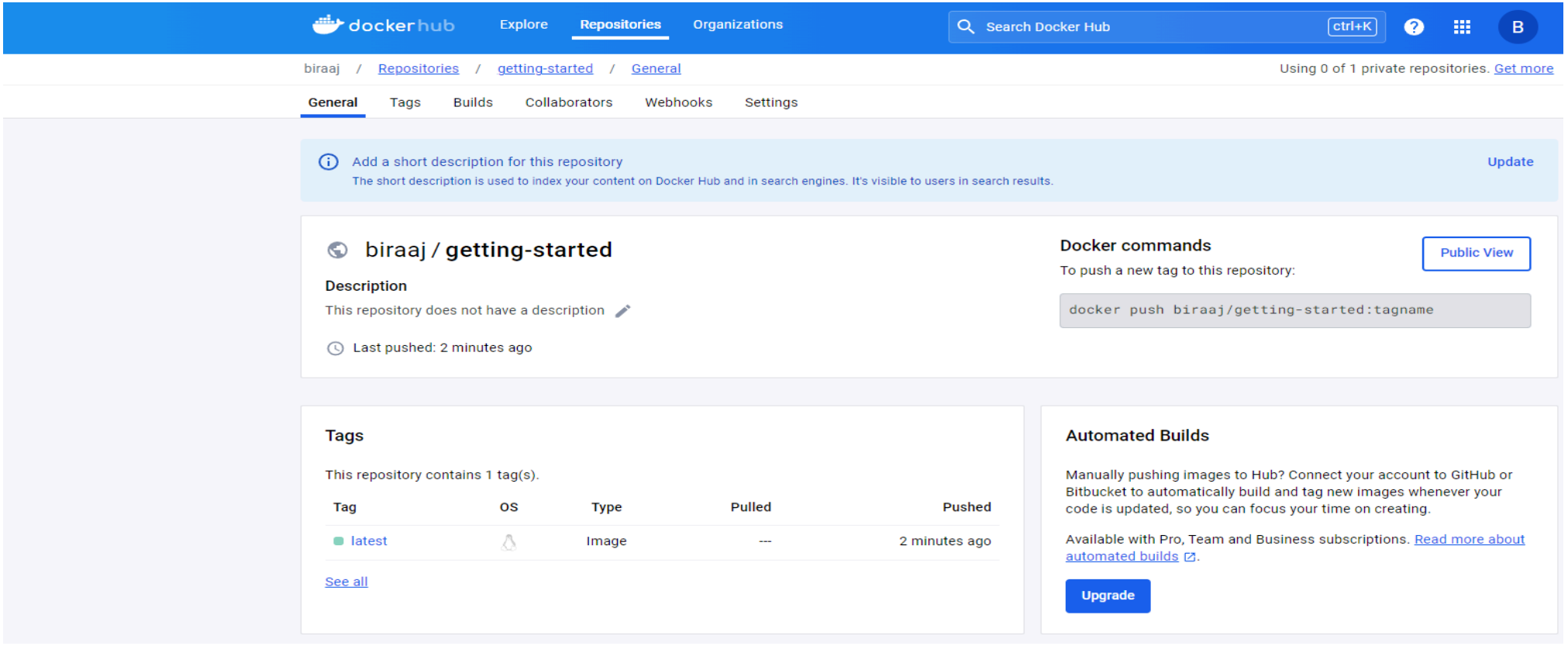
i. **Link to view the docker image**: https://hub.docker.com/repository/docker/biraaj/getting-started
ii. Signed in to Docker hub
iii. Created a repository with a name “getting-started”
iv. Pushed the image
    1. Signed in to the docker hub in cli.



2. Use the docker tag and Docker push to push the image into the newly created repository
    a. **docker login -u biraaj;**
        i. This command signs in to docker hub in cli. Here the username is biraaj and it will ask password after executing this command.

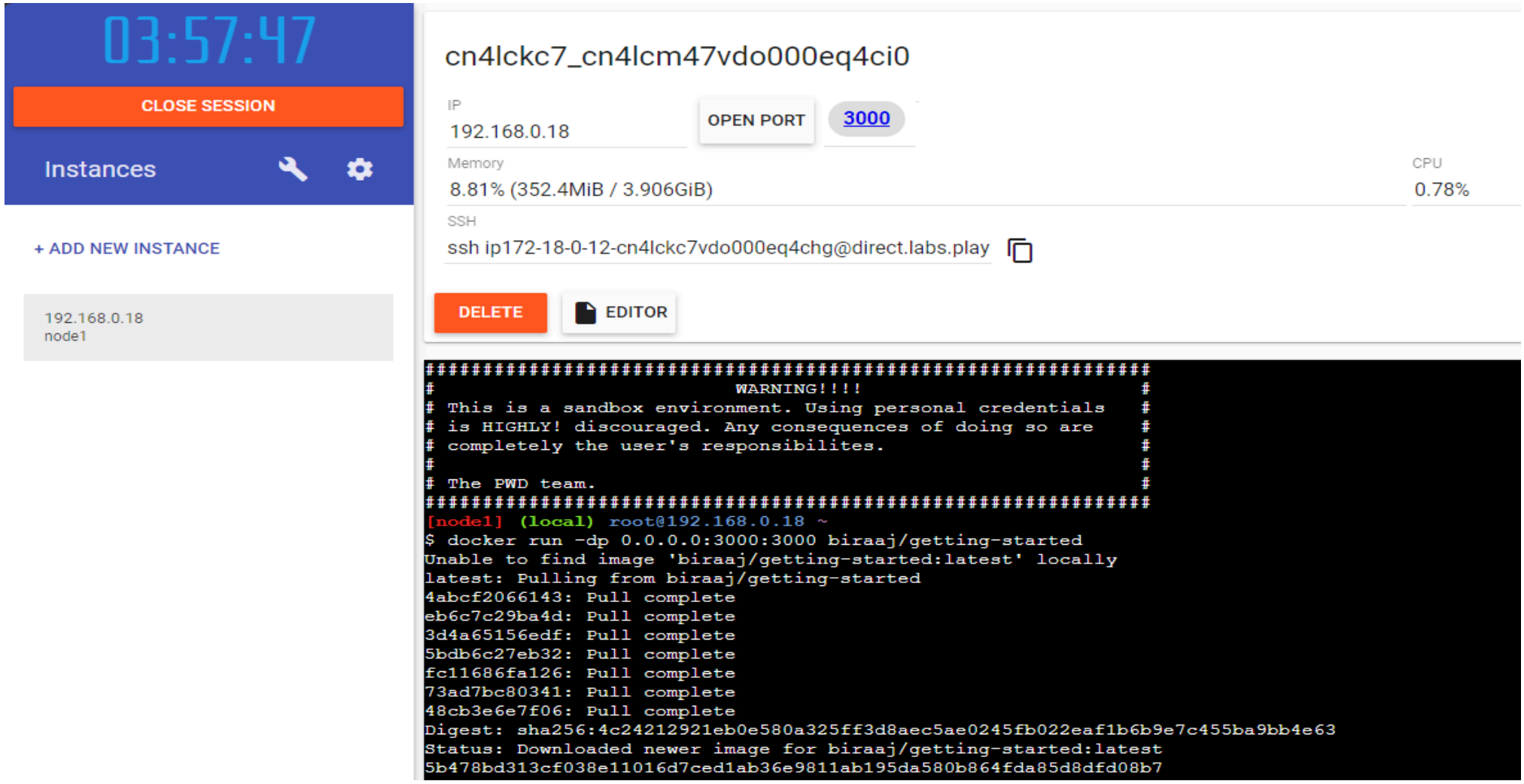
b. **docker tag getting-started biraaj/getting-started:**
    i. This command gives a new name to the image.
c. **docker push biraaj/getting-started**
    i. This command pushes the newly created image to the repository mentioned at the end here.

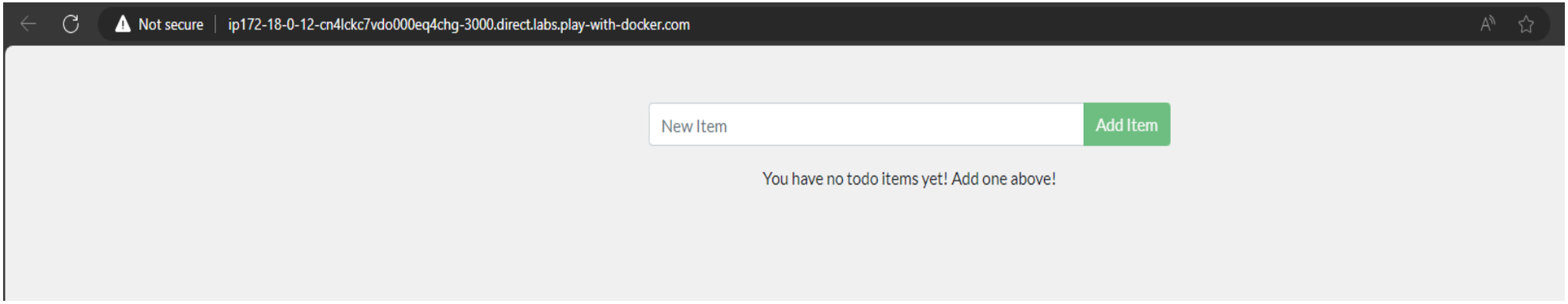




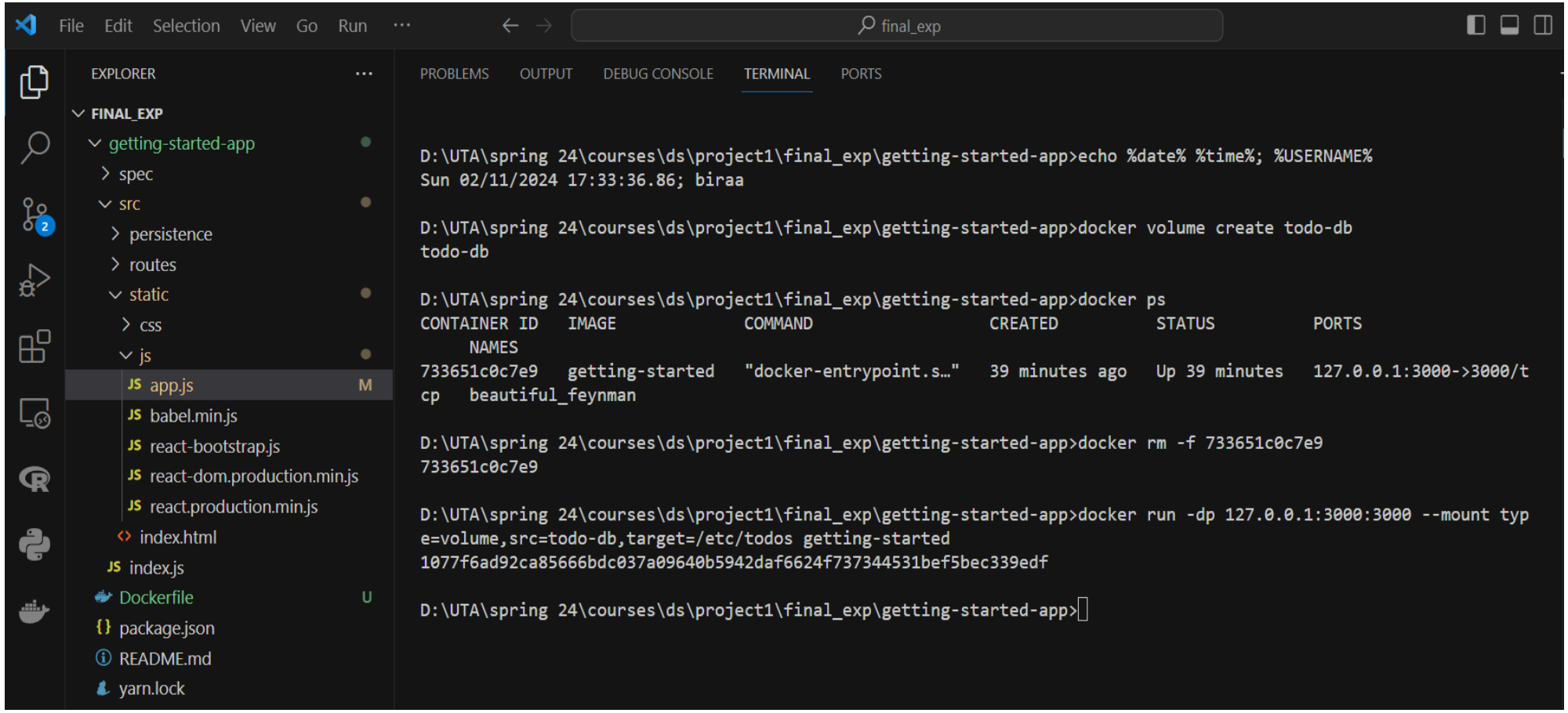
3. We can see the latest tag as image is pushed to docker hub. As we didn't add a tag name it mentioned it as latest.

v. Ran the Image on a new instance to verify that it runs on different machine.
   1. opened Play with Docker and logged into the users(biraaj) docker account.
   2. Selected the ADD NEW INSTANCE
   3. In terminal ran the freshly pushed docker app


03:57:47
CLOSE SESSION
Instances
+ ADD NEW INSTANCE
192.168.0.18
node1
cn4lckc7_cn4lcm47vdo000eq4ci0
IP
192.168.0.18
OPEN PORT
3000
Memory
8.81% (352.4MiB / 3.906GiB)
CPU
0.78%
SSH
ssh ip172-18-0-12-cn4lckc7vdo000eq4chg@direct.labs.play
DELETE
EDITOR
WARNING!!!!
This is a sandbox environment. Using personal credentials
is HIGHLY! discouraged. Any consequences of doing so are
completely the user's responsibilites.
The PWD team.
[node1] (local) root@192.168.0.18 ~
$ docker run -dp 0.0.0.0:3000:3000 biraaj/getting-started
Unable to find image 'biraaj/getting-started:latest' locally
latest: Pulling from biraaj/getting-started
4abcf2066143: Pull complete
eb6c7c29ba4d: Pull complete
3d4a65156edf: Pull complete
5bdb6c27eb32: Pull complete
fc11686fa126: Pull complete
73ad7bc80341: Pull complete
48cb3e6e7f06: Pull complete
Digest: sha256:4c24212921eb0e580a325ff3d8aec5ae0245fb022eaf1b6b9e7c455ba9bb4e63
Status: Downloaded newer image for biraaj/getting-started:latest
5b478bd313cf038e11016d7ced1ab36e9811ab195da580b864fda85d8dfd08b7

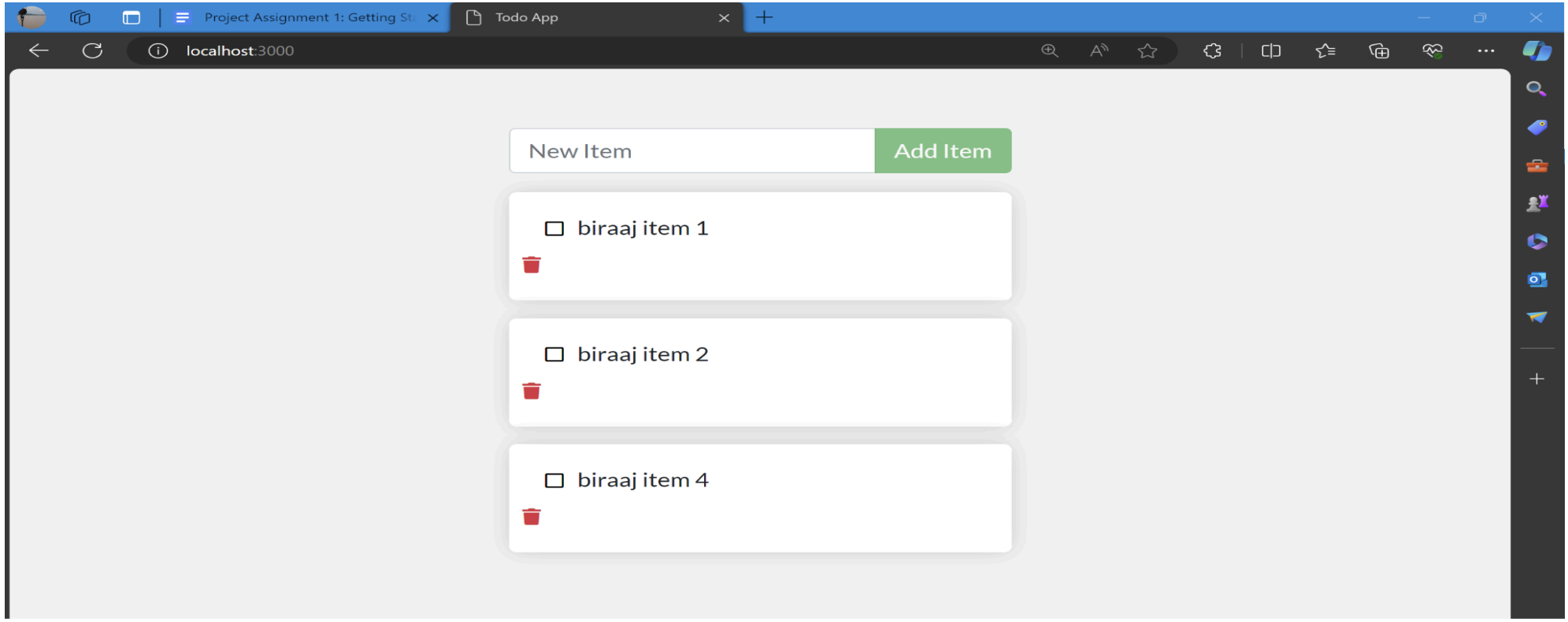


f. Part 5: persist the DB
   i. Created a volume using docker volume command
      1. Command: **docker volume create todo-db**
         a. This command is used to create volumes which is todo-db here.
         b. Volumes provide the ability to connect specific filesystem paths of the container back to the host machine
   ii. Stoped and remove the todo app container using **docker rm -f <id>**
      1. As we know that docker rm is used to remove containers as explained in part the -f options forcefully stops the container and removes it.
   iii. Started the todo app container using "**docker run -dp 127.0.0.1:3000:3000 --mount type=volume,src=todo-db,target=/etc/todos getting-started**"(explained in part2) , but added the **--mount** option to specify a volume mount. Gave the volume a name, and mounted it to **/etc/todos** in the container.

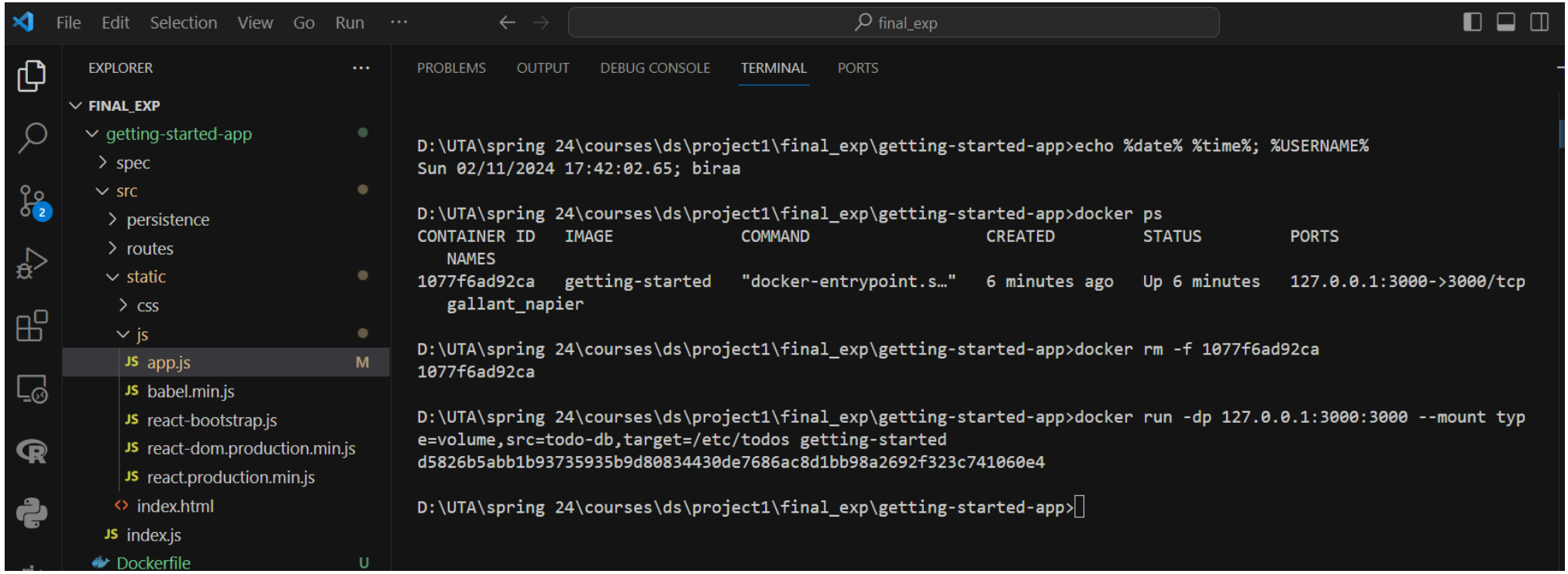


iv. Verify that the data persists.

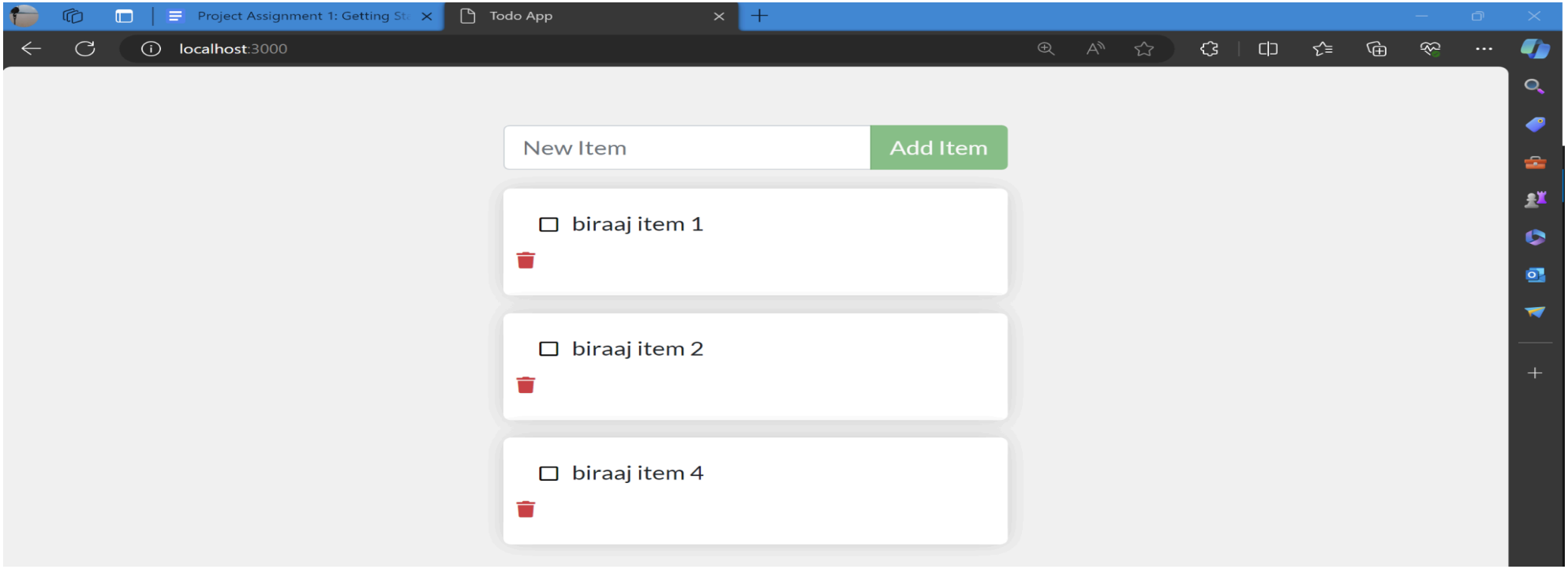
1. Once the container started we added some items to the application.



2. Stopped the container using **docker rm -f <id>**(explained in part2) and started a new container.

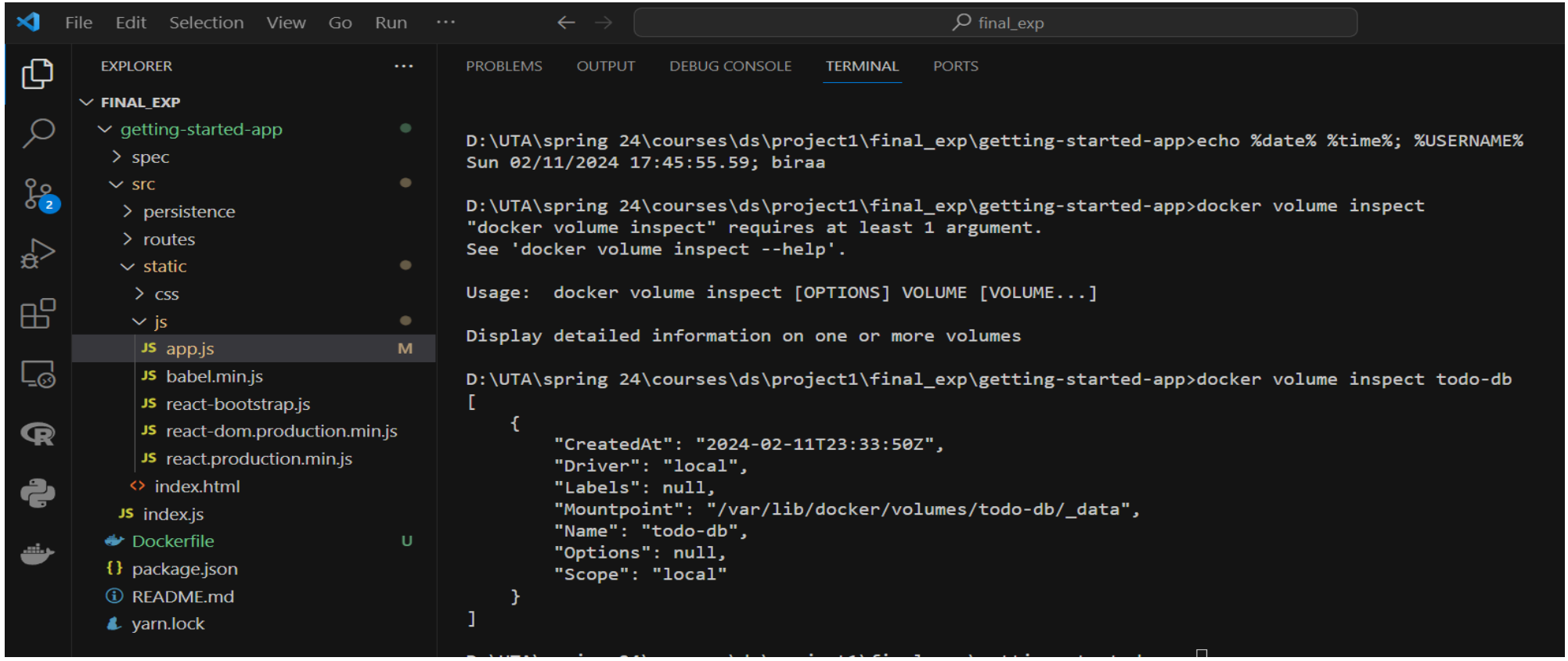


3. Checking if the data persists by refreshing the app. We see that same data is visible again after starting the container from scratch again.



v. Checked the volume info

1. The command **docker volume inspect** is used to see where the docker is storing data in host filesystem.



g. Part 6: Using bind mounts

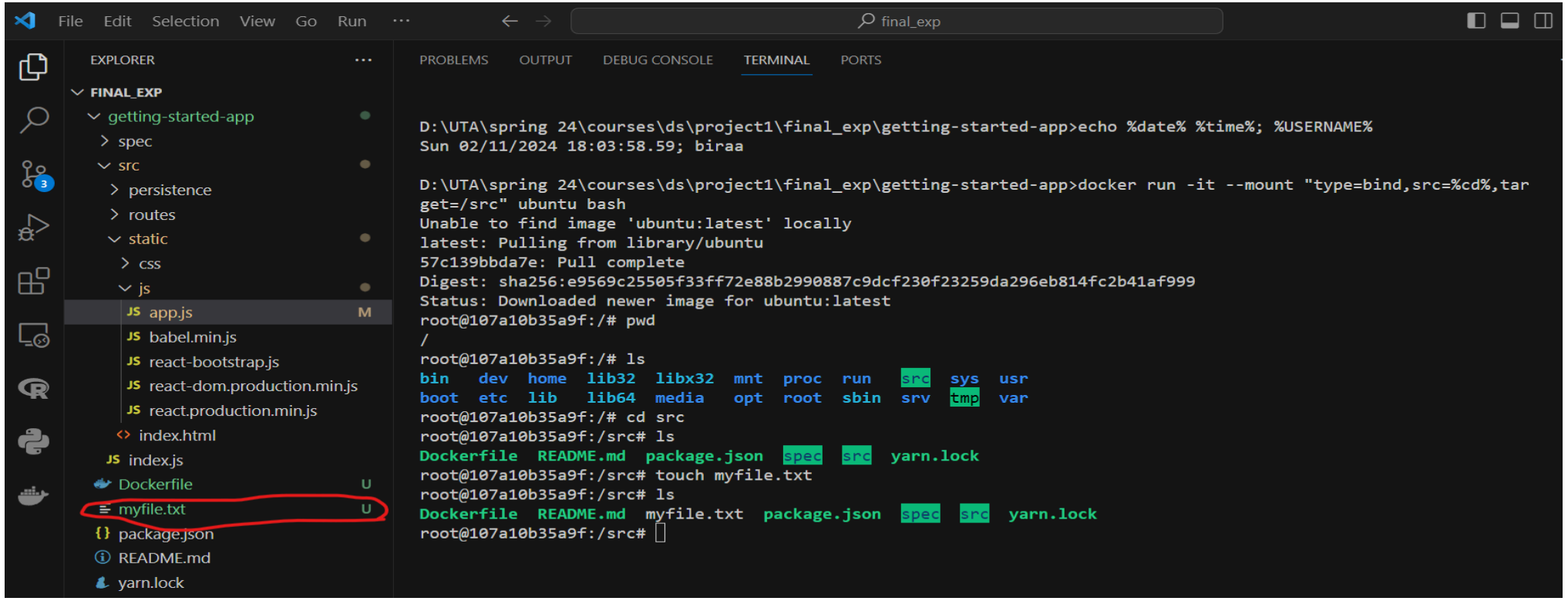
i. Trying the bind mounts

1. Start bash in an Ubuntu container

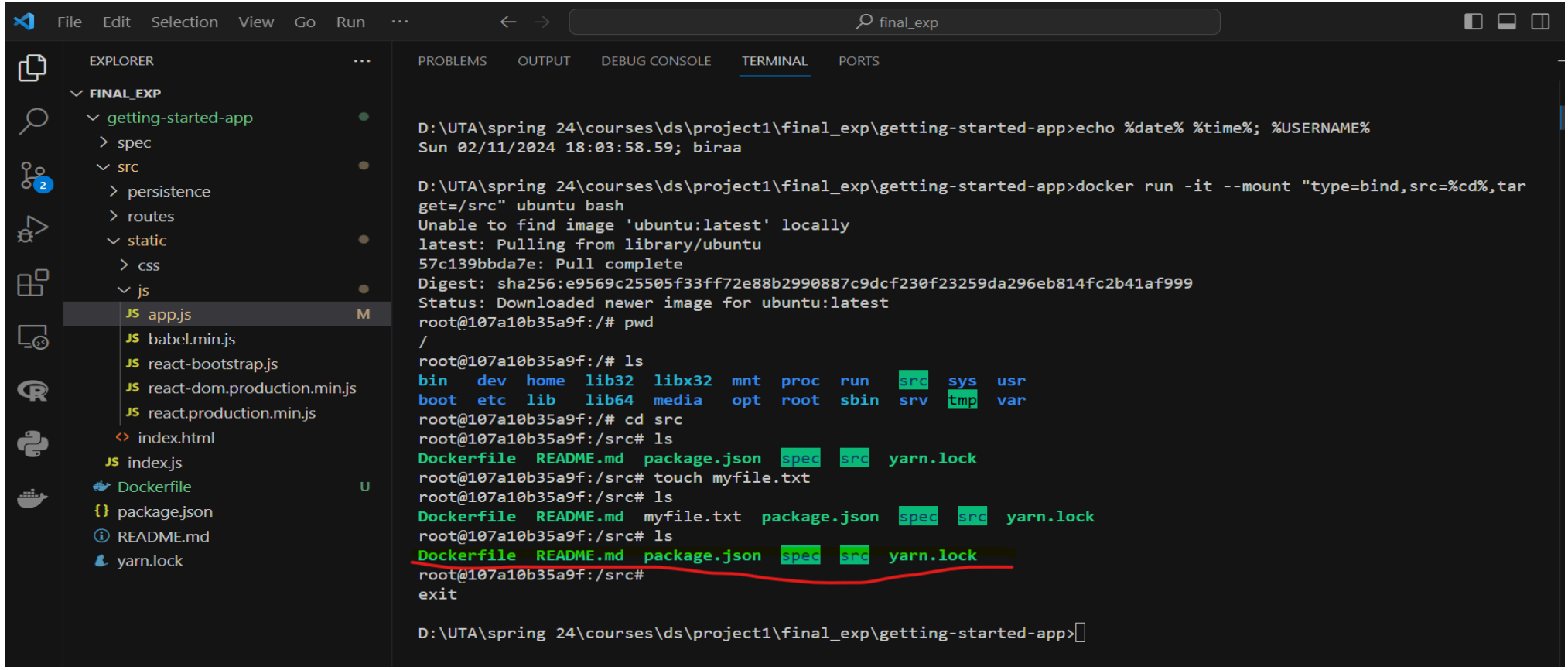
a. Command: **docker run -it --mount "type=bind,src=%cd%,target=/src" ubuntu bash**

i. This run command with mount is explained in part 5 and here we added extra parameters src ,target ubuntu and bash were src is current working directory, target is where the directory should appear inside the container and ubuntu bash starts the bash in terminal.

2. Docker starts an interactive bash session in the root directory of the container.
3. Created a file called **myfile.txt** inside source which is now observed in the getting-started-app directory



4. Deleting the file from the host we saw that the file was gone from the container too and we stopped all interactive sessions using **ctrl+D**

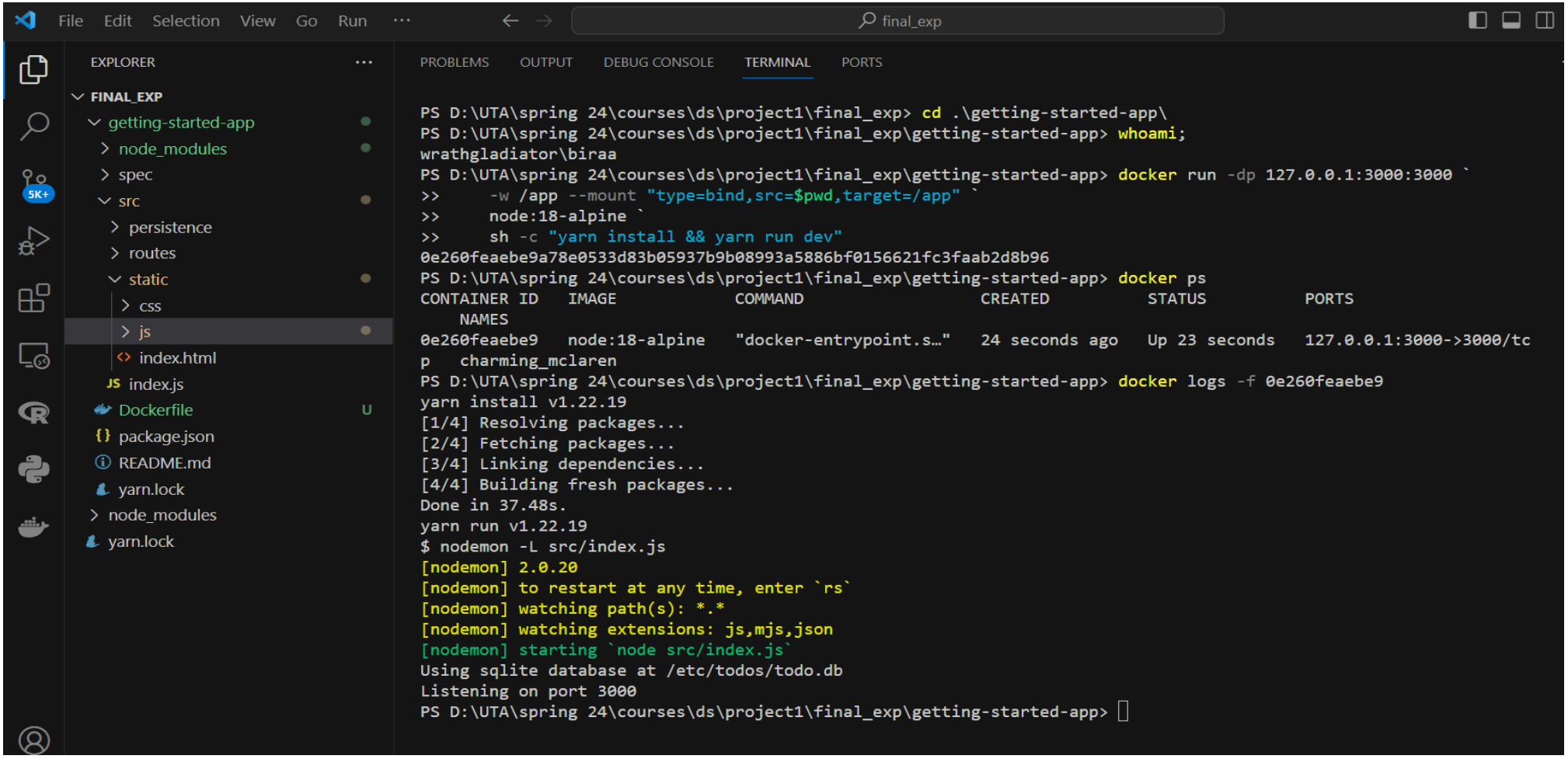


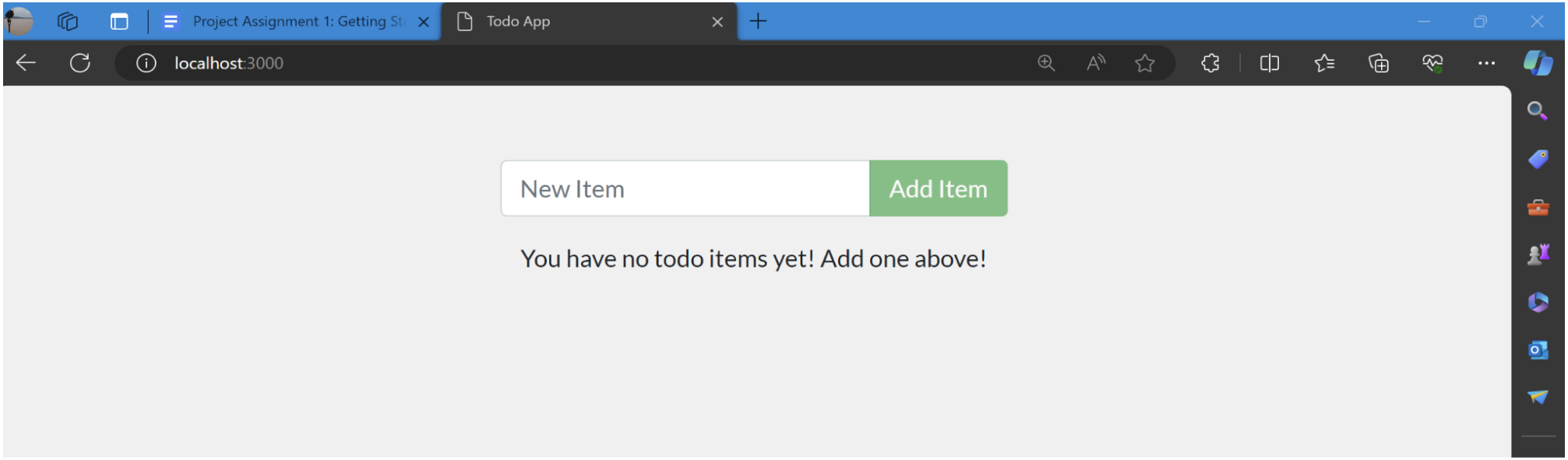
ii. Development containers

1. Removed all the running containers of getting-started app using the command docker **rm -f <container-id>**
2. Then Ran the command below to initiate a development container with **nodemon**
    a. Command: **docker run -dp 127.0.0.1:3000:3000 -w /app --mount "type=bind,src=$pwd,target=/app" node:18-alpine sh -c "yarn install && yarn run dev"**

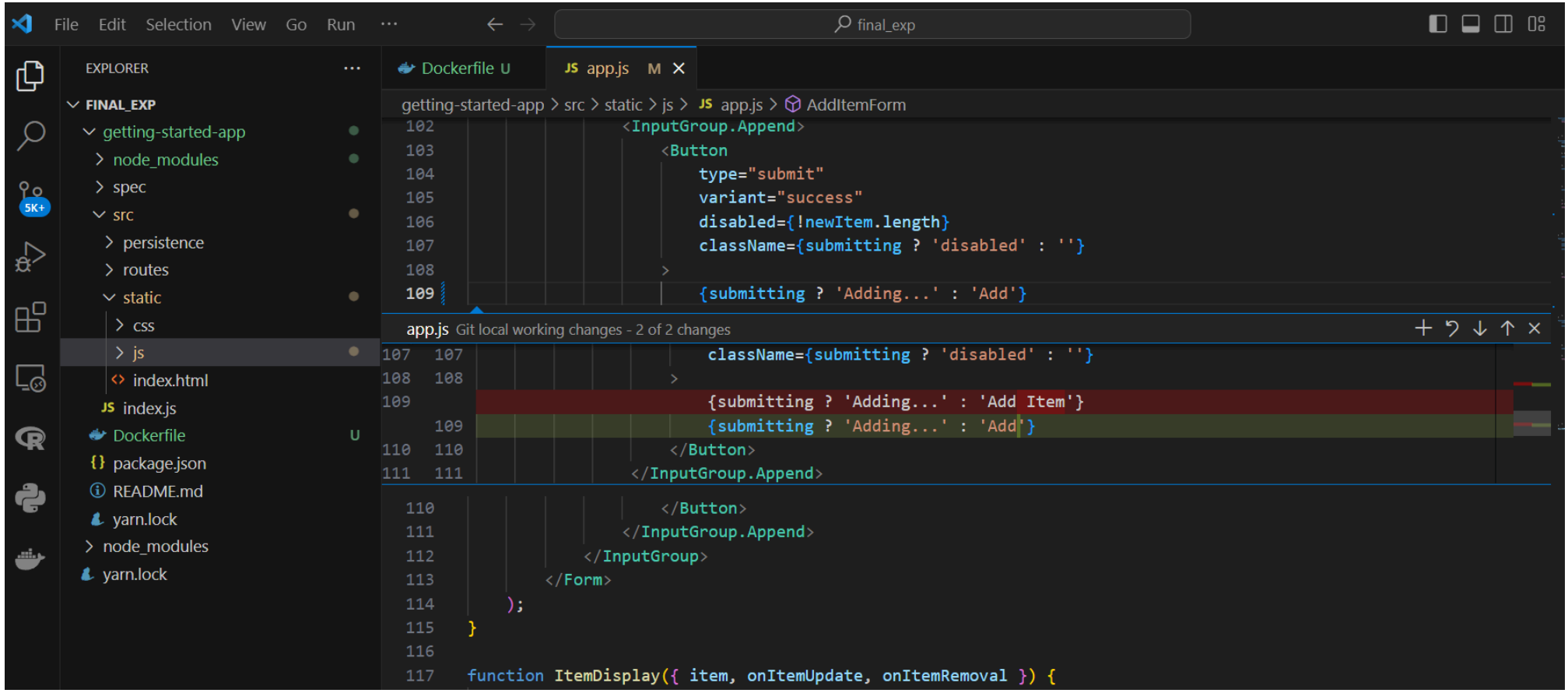
    i. The command is well explained in part2 and 3 . so here **-w /app** - sets the "working directory" or the current directory that the command will run from. **--mount "type=bind,src=$pwd,target=/app"**,bind mount the current directory from the host into the /app directory in the container. **node:18-alpine** - the image to use. Note that this is the base image for app from the Dockerfile. **sh -c "yarn install && yarn run dev"** - the command. It is starting a shell using sh (alpine doesn't have bash) and running yarn install to install packages and then running yarn run dev to start the development server. If we look in the package.json, we will see that the dev script starts nodemon.

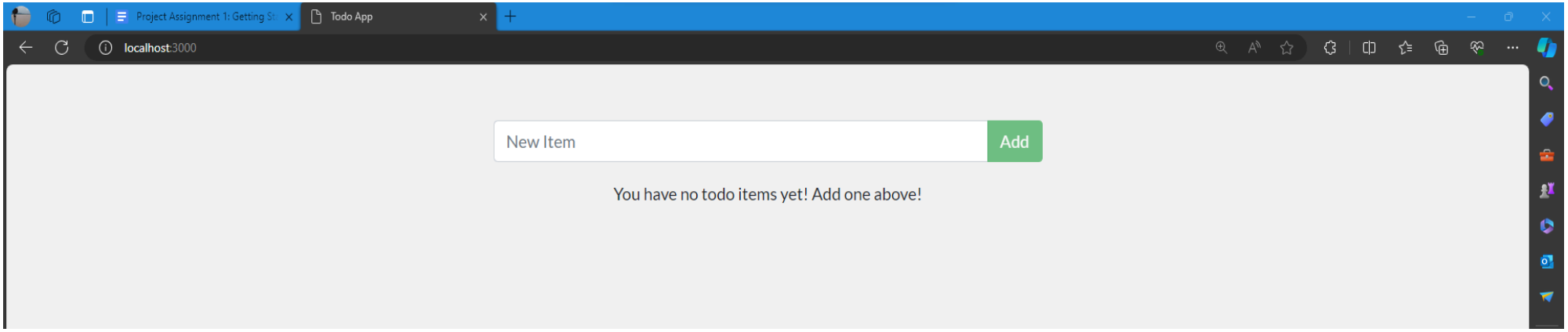
b. Watch logs using the command **docker logs <container-id>**
    i. This command is used to see logs for docker run as we see here the starting of nodemon and sqlite db.


File Edit Selection View Go Run ···
final_exp
EXPLORER
FINAL_EXP
getting-started-app
node_modules
spec
src
persistence
routes
static
css
js
index.html
index.js
Dockerfile
package.json
README.md
yarn.lock
node_modules
yarn.lock
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\UTA\spring 24\courses\ds\project1\final_exp> cd .\getting-started-app\
PS D:\UTA\spring 24\courses\ds\project1\final_exp\getting-started-app> whoami;
wrathgladiator\biraa
PS D:\UTA\spring 24\courses\ds\project1\final_exp\getting-started-app> docker run -dp 127.0.0.1:3000:3000 `
>> -w /app --mount "type=bind,src=$pwd,target=/app" `
>> node:18-alpine `
>> sh -c "yarn install && yarn run dev"
0e260feaebe9a78e0533d83b05937b9b08993a5886bf0156621fc3faab2d8b96
PS D:\UTA\spring 24\courses\ds\project1\final_exp\getting-started-app> docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
0e260feaebe9 node:18-alpine "docker-entrypoint.s…" 24 seconds ago Up 23 seconds 127.0.0.1:3000->3000/tc
p charming_mclaren
PS D:\UTA\spring 24\courses\ds\project1\final_exp\getting-started-app> docker logs -f 0e260feaebe9
yarn install v1.22.19
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...
Done in 37.48s.
yarn run v1.22.19
$ nodemon -L src/index.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node src/index.js`
Using sqlite database at /etc/todos/todo.db
Listening on port 3000
PS D:\UTA\spring 24\courses\ds\project1\final_exp\getting-started-app>



c. After updating the **app.js** the change is reflected without stopping the container or building a new image.

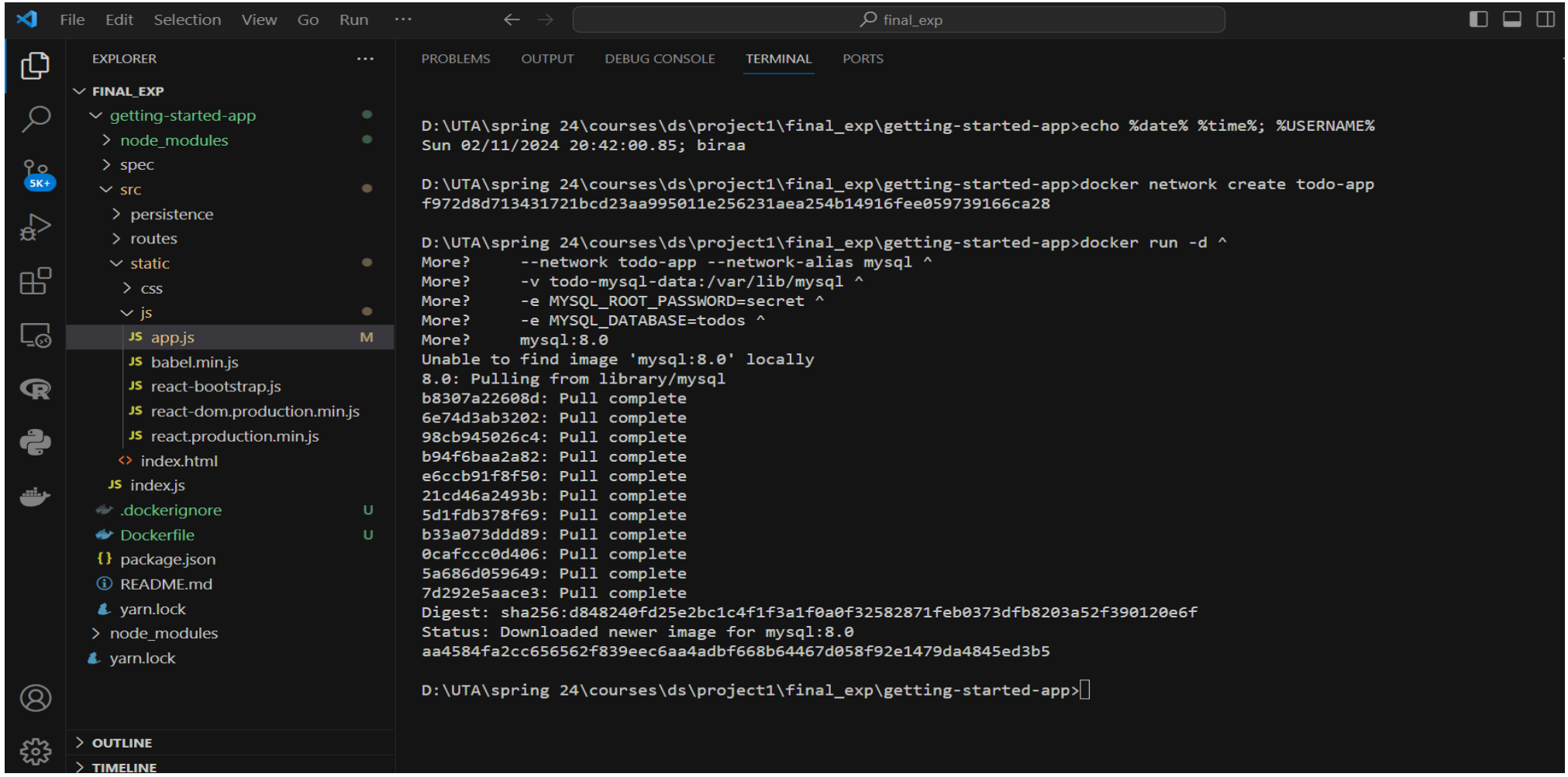

File Edit Selection View Go Run ···
final_exp
EXPLORER
FINAL_EXP
getting-started-app
node_modules
spec
src
persistence
routes
static
css
js
index.html
index.js
Dockerfile U
package.json
README.md
yarn.lock
node_modules
yarn.lock
Dockerfile U
app.js M
getting-started-app > src > static > js > app.js > AddItemForm
<InputGroup.Append>
<Button
type="submit"
variant="success"
disabled={!newItem.length}
className={submitting ? 'disabled' : ''}
>
{submitting ? 'Adding...' : 'Add'}
app.js Git local working changes - 2 of 2 changes
className={submitting ? 'disabled' : ''}
>
{submitting ? 'Adding...' : 'Add Item'}
{submitting ? 'Adding...' : 'Add'}
</Button>
</InputGroup.Append>
</Button>
</InputGroup.Append>
</InputGroup>
</Form>
);
}
function ItemDisplay({ item, onItemUpdate, onItemRemoval }) {



h. Part7: Multi container apps

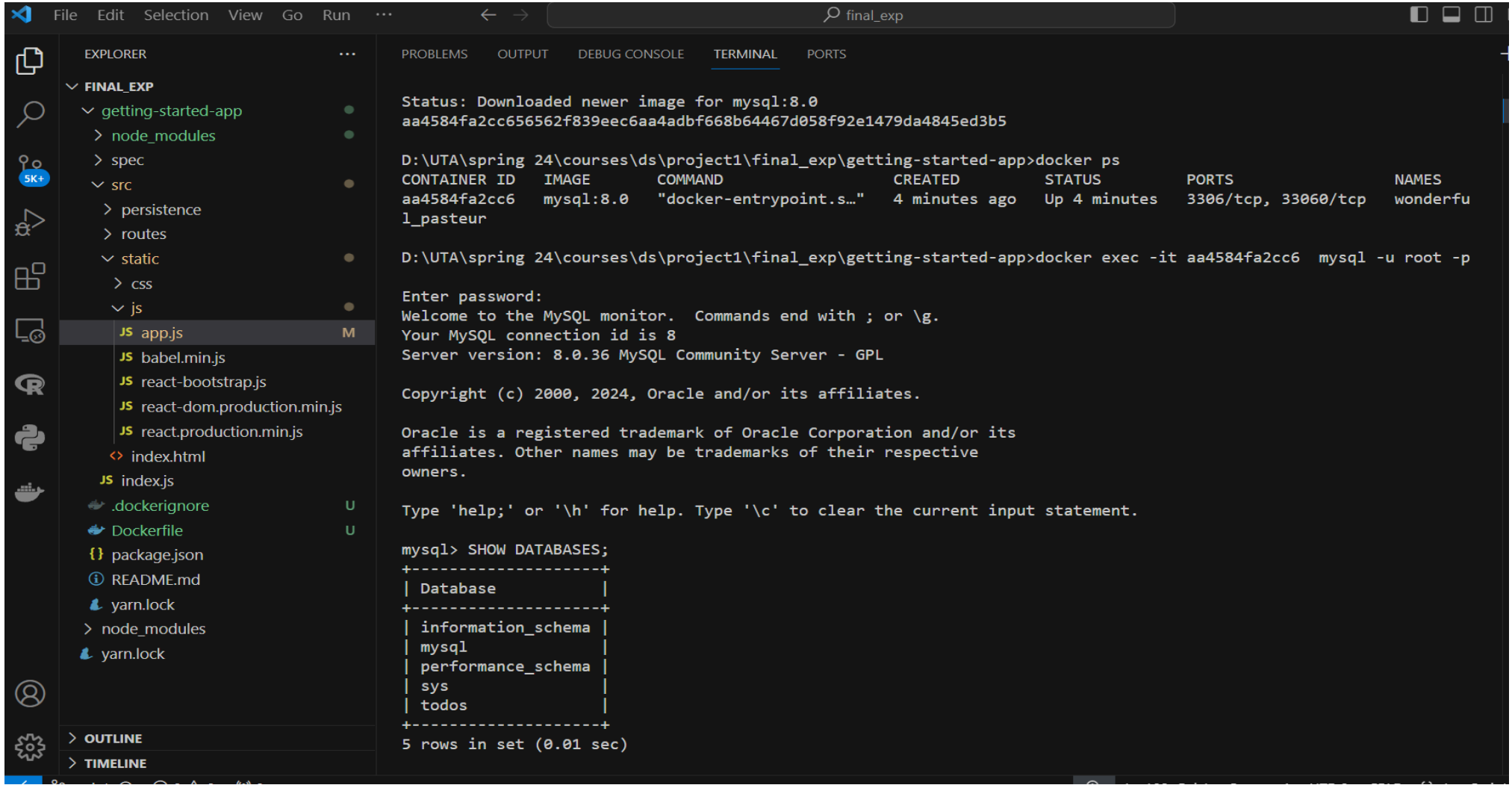
i. Created a container network using command “**docker network create todo-app**”:
   1. Here a network called todo-app is created using the above command.

ii. Ran MySQL container and attached it to the created network
   1. **docker run -d --network todo-app --network-alias mysql -v todo-mysql-data:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=secret -e MYSQL_DATABASE=todos mysql:8.0**
      a. This Docker command runs a MySQL container in detached mode (-d) with the specified configurations:
      b. It connects the container to a Docker network named "todo-app" with the alias "mysql".
      c. It mounts a Docker volume named "todo-mysql-data" to persist MySQL data.
      d. Sets the MySQL root password to "secret" and creates a database named "todos".
      e. It uses the MySQL 8.0 image to create the container.
   2. **docker exec -it <mysql-container-id> mysql -u root -p**
      a. This Docker command allows interactive access (-it) to a running MySQL container (<mysql-container-id>) as the root user (-u root) with a prompt for the password (-p).



iii. Checked if the database was running using the commands below.

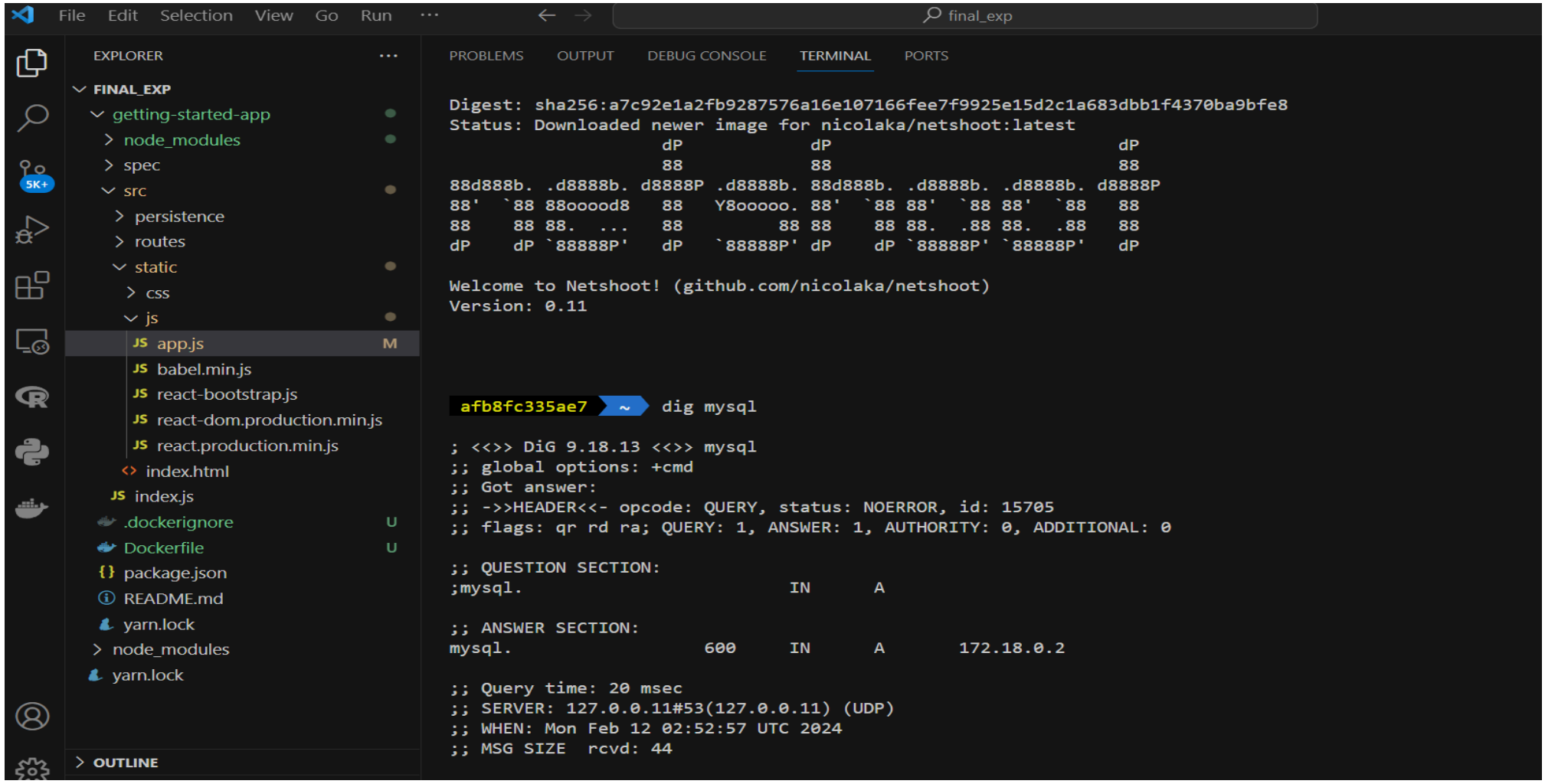
1. **mysql> SHOW DATABASES;**
    a. This shows all the available databases.



iv. Used **nicolaka/netshoot** container to troubleshoot networking issues.

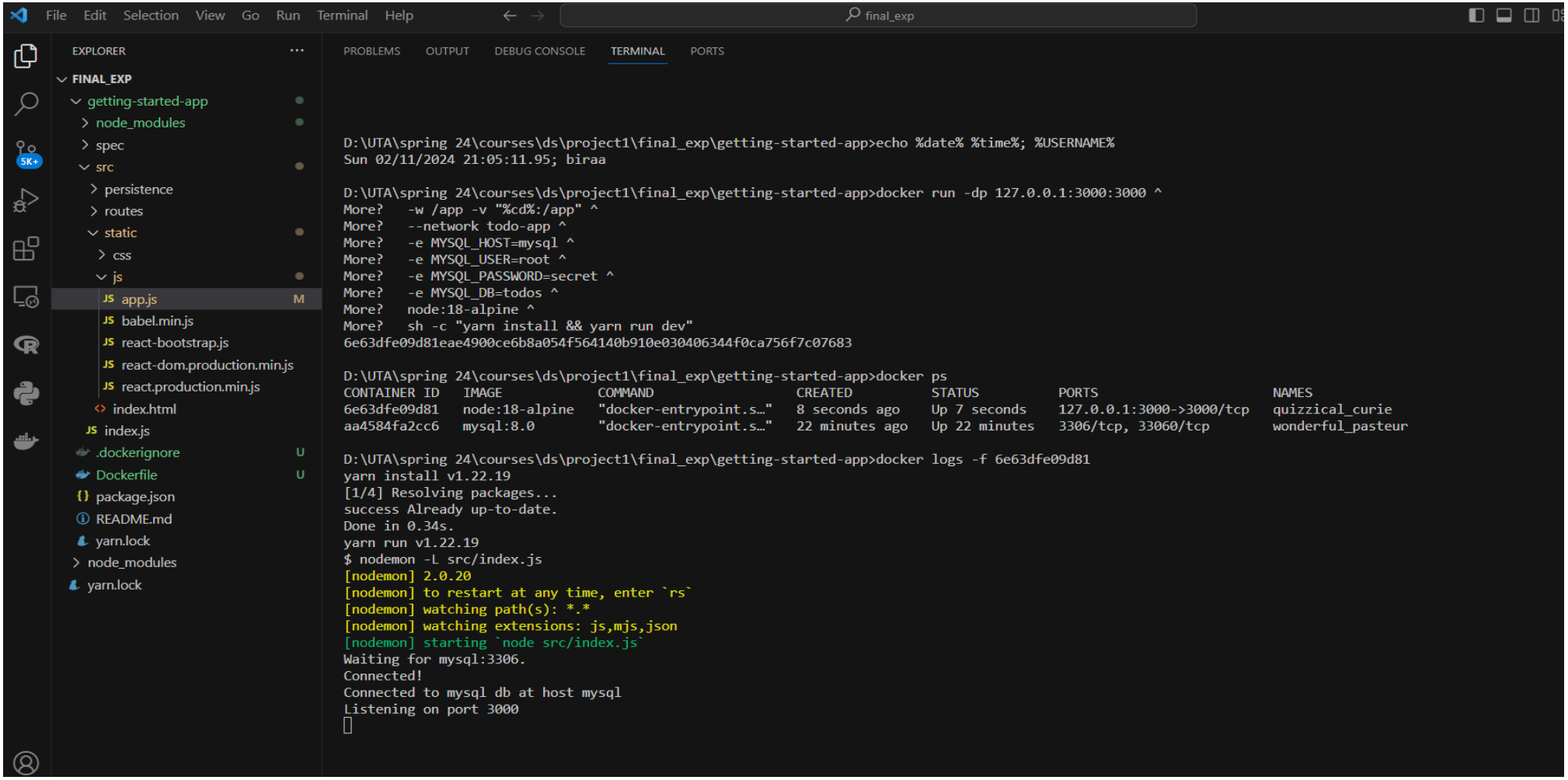
1. Command1: **docker run -it --network todo-app nicolaka/netshoot**

    a. This Docker command runs the "netshoot" container interactively (-it) within the "todo-app" Docker network, providing a toolbox of networking utilities for troubleshooting and diagnostics.
2. Command2: **dig mysql**
    a. This command uses dig to perform a DNS lookup for the hostname "mysql".
3. In the "ANSWER SECTION", we see an A record for mysql that resolves to 172.18.0.2 . While mysql isn't normally a valid hostname, Docker was able to resolve it to the IP address of the container that had that network alias as we used the **--network-alias** earlier.

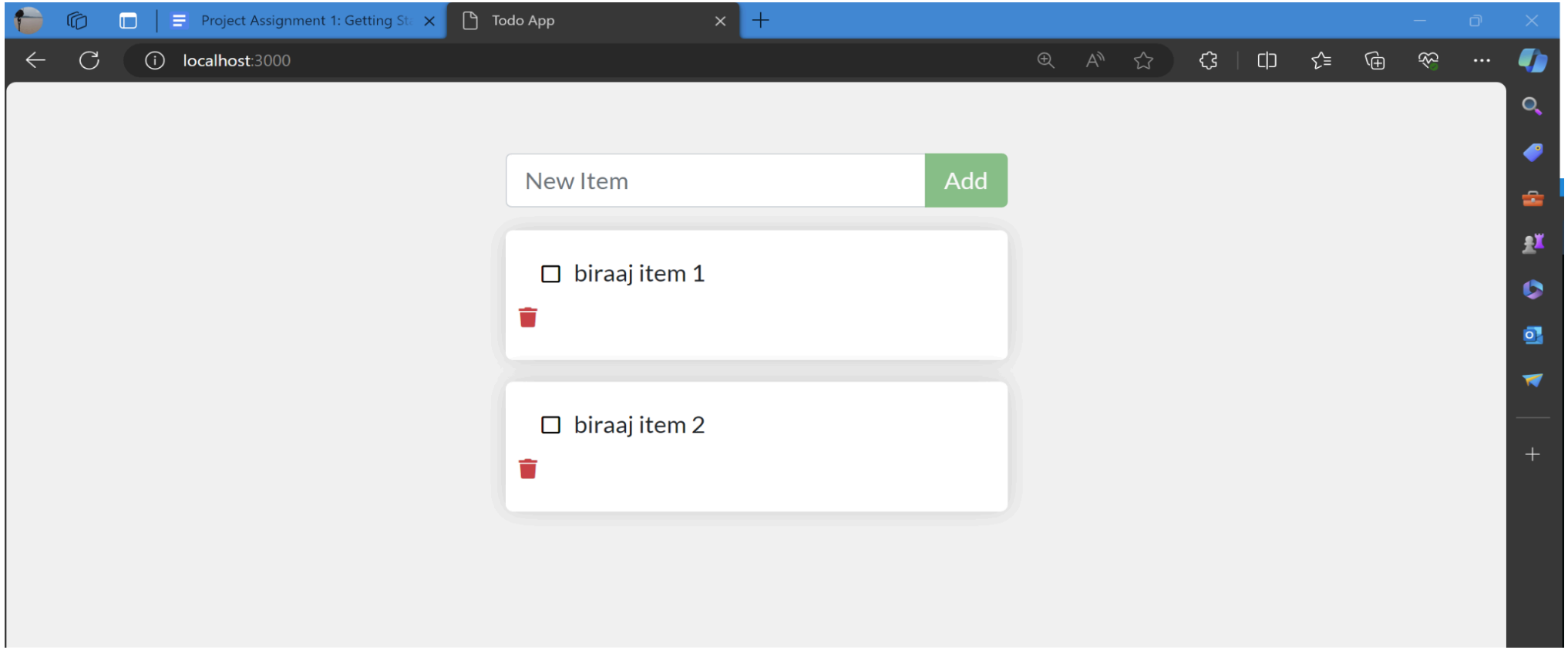


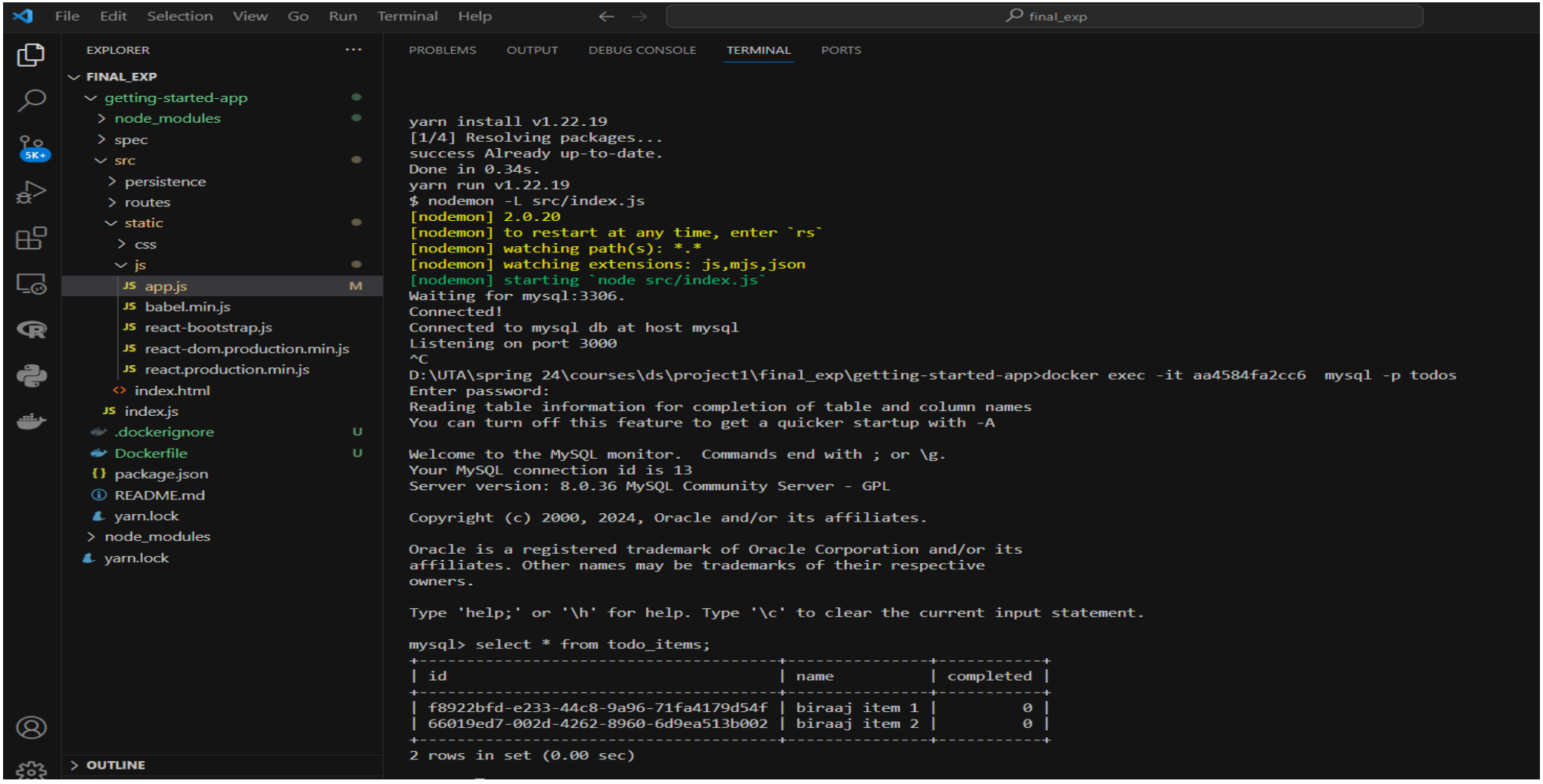
4. Started the todo app with MySQL environment variables using the commands below:
    a. **docker run -dp 127.0.0.1:3000:3000 -w /app -v "%cd%:/app" --network todo-app -e MYSQL_HOST=mysql -e MYSQL_USER=root -e MYSQL_PASSWORD=secret -e MYSQL_DB=todos node:18-alpine sh -c "yarn install && yarn run dev"**

i. This Docker command runs a Node.js application in detached mode (-d), mapping port 3000 on localhost (127.0.0.1:3000:3000), using the "todo-app" network, and setting environment variables for MySQL connection. It mounts the current directory (%cd%) to /app in the container, sets the working directory to /app, and executes the commands to install dependencies and runs the development server.

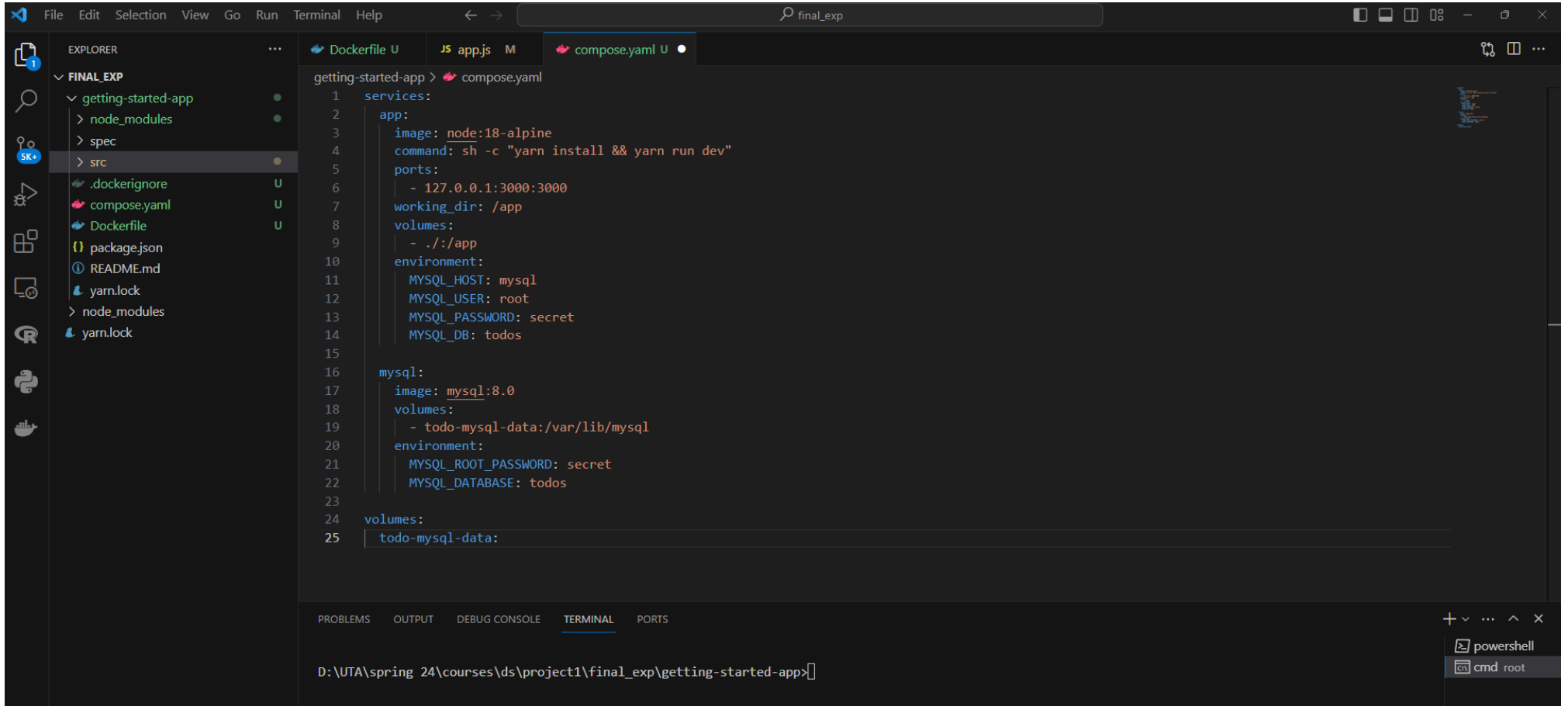


5. Adding items to the todo list and checking them in the MySQL database.


Project Assignment 1: Getting St
Todo App
localhost:3000
New Item
Add
biraaj item 1
biraaj item 2



i. Part 8: using docker-compose
    i. Created a docker-compose file with the following content

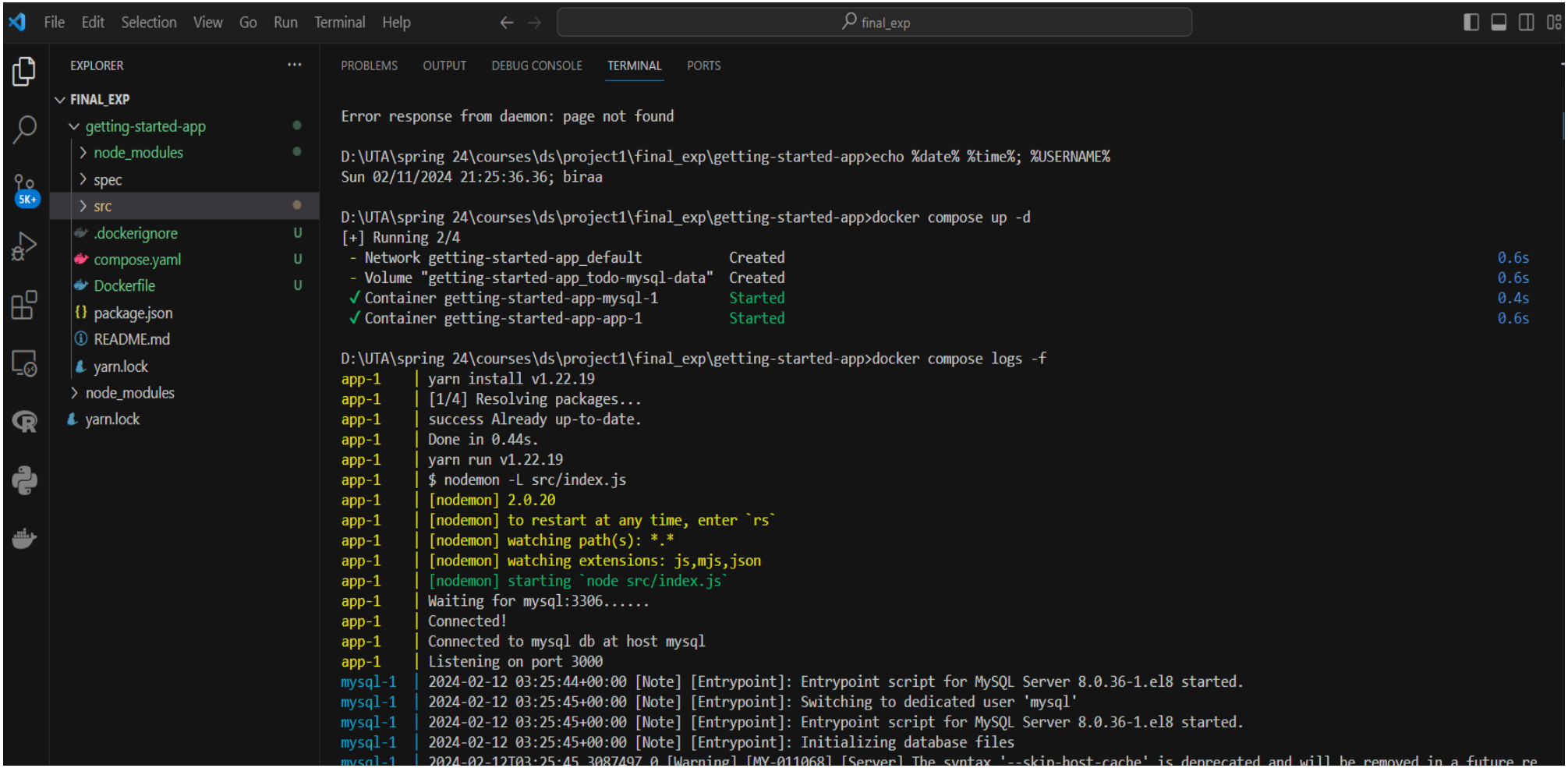


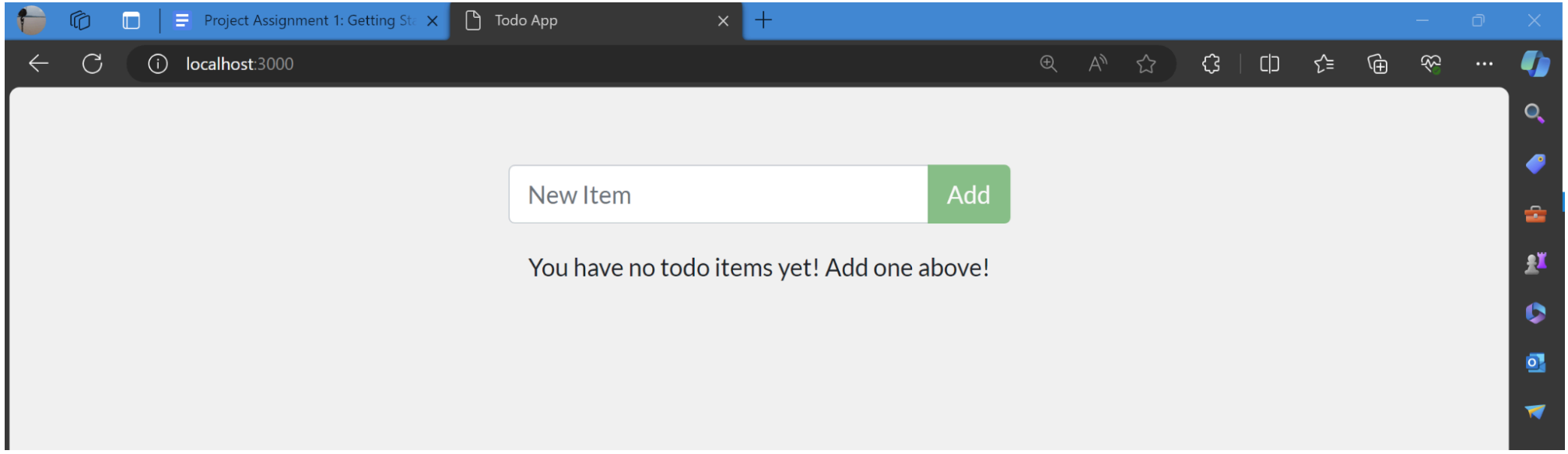
ii. Deleted all running containers using "**docker rm -f $(docker ps -aq)**" to have uninterrupted run. Here **docker ps -aq** lists all the docker container ids that are running.

iii. Run docker-compose as below and verify it in the logs

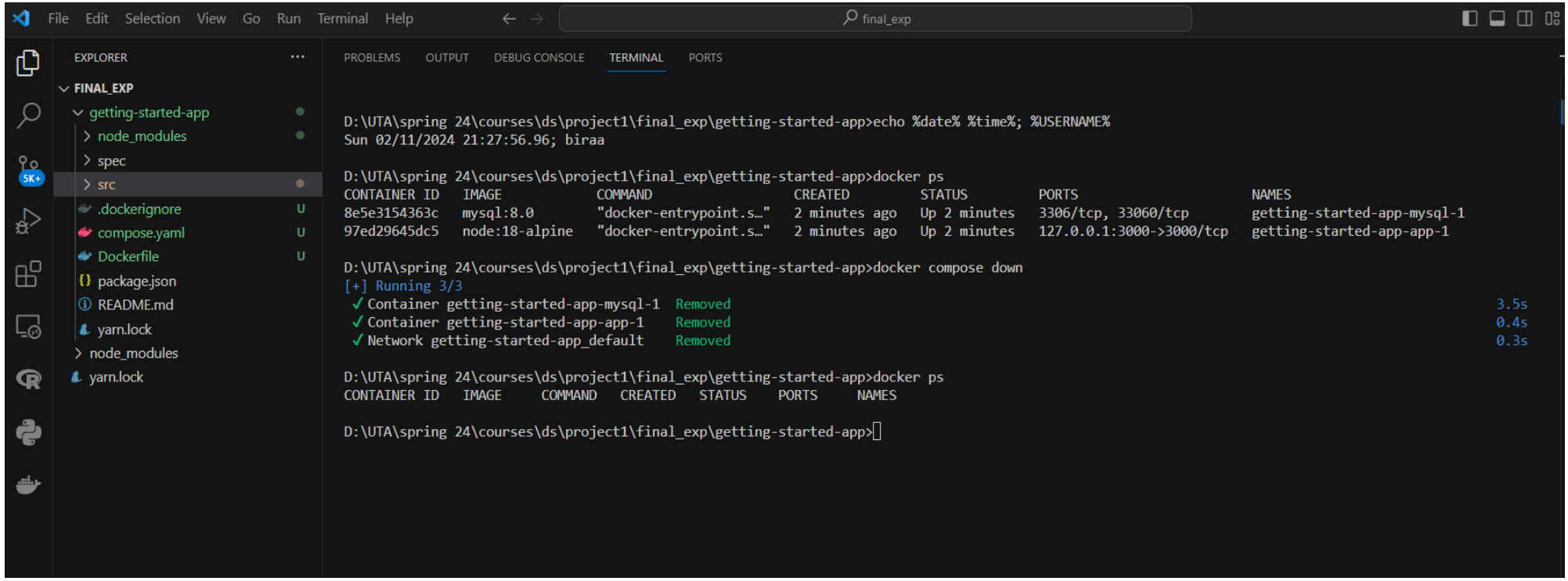
1. Command: **docker compose up -d**
    a. Starts up the application stack as mentioned in docker compose file.
    b. Command: **docker compose logs -f**

i. This shows logs from each of the services interleaved into a single stream. The -f flag follows the log, so will give the live output as it's generated.
ii. **docker compose logs -f app** helps in viewing the logs for specific service as required.

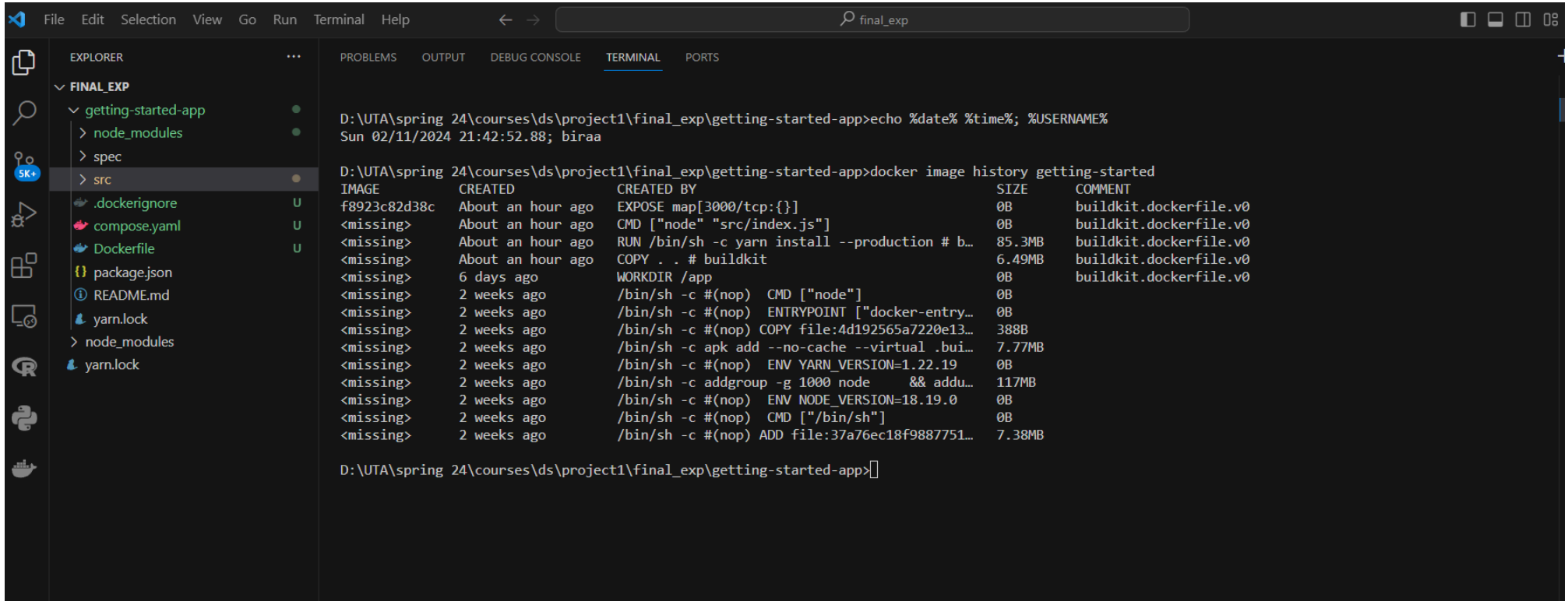




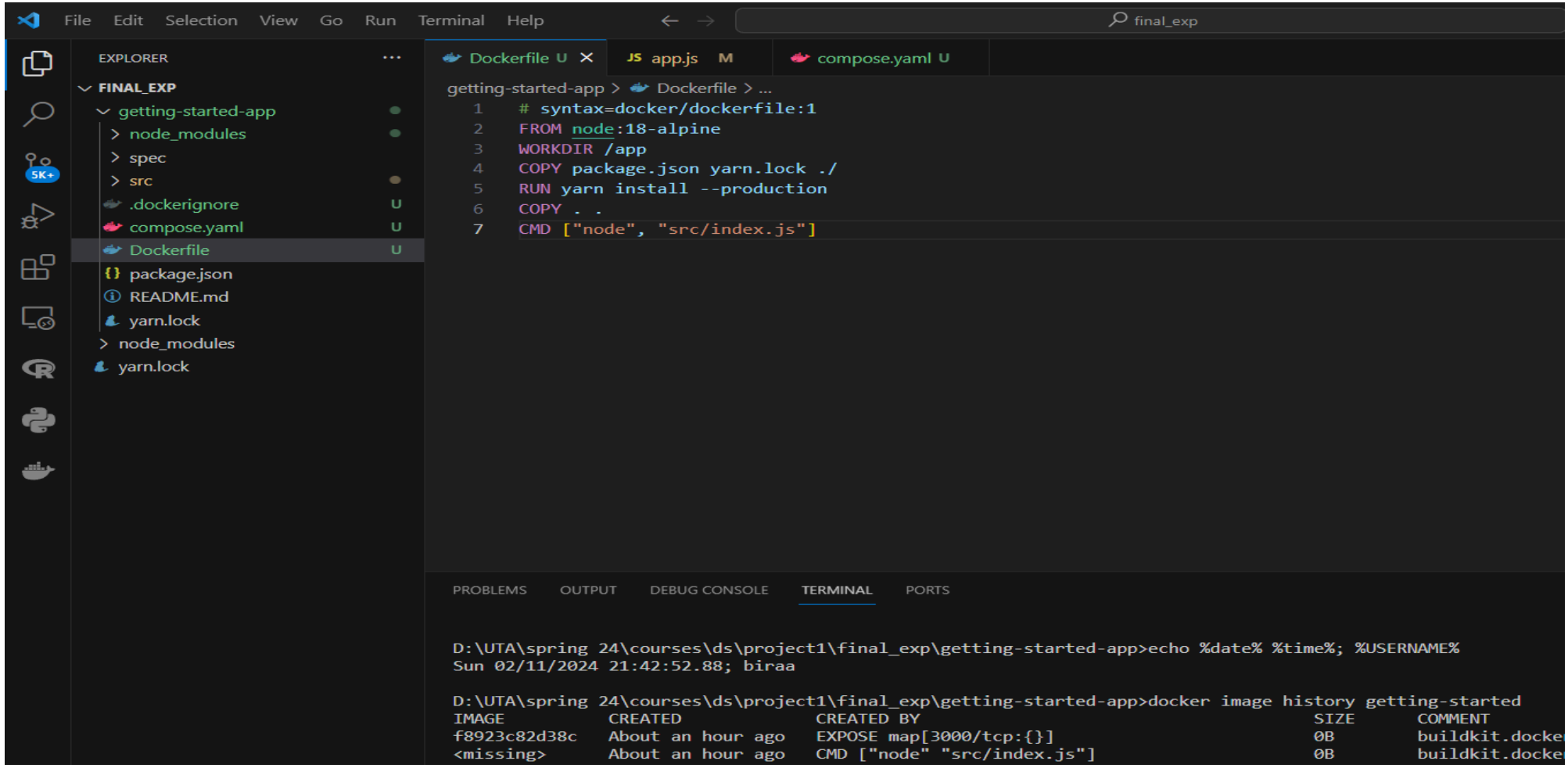
iv. Destroying all containers using “**docker compose down**”. It kills all services that are running in the application.



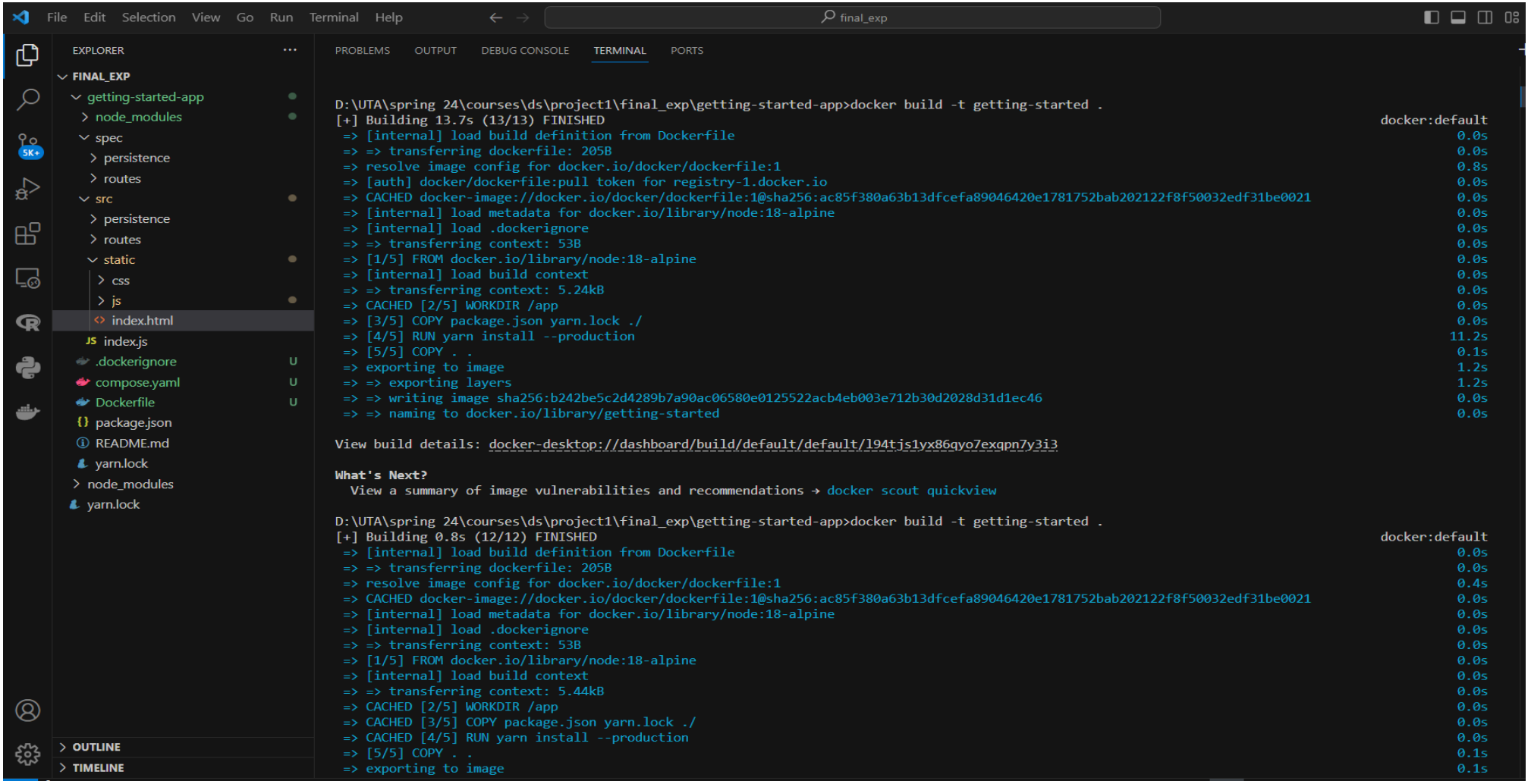
j. Part 9: image building best practices
   i. Image layering
      1. Checking history using **docker image history.** Here it shows the layers in the getting-started image.



2. Updating Dockerfile to decrease build time for which we need to support caching of dependencies. So here copying **package.json** first will help stop reinstalling yarn dependencies.



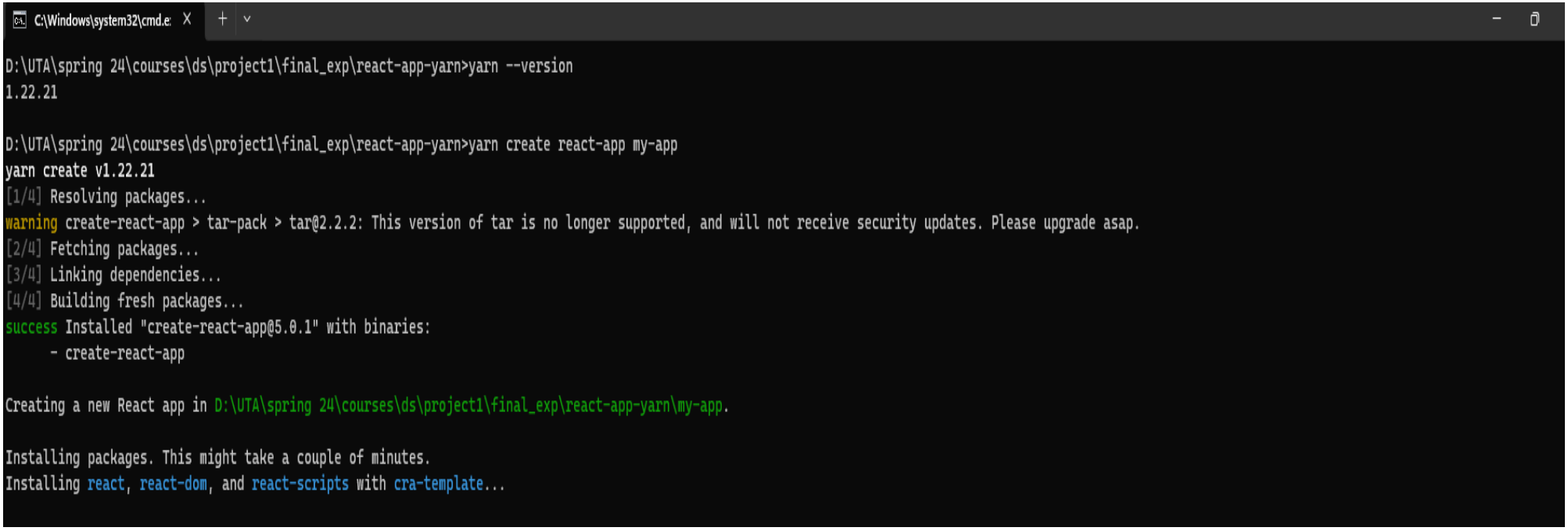
3. Used .**dockerignore** file to copy only relevant image files by excluding **node_modules**
4. Built the getting started image once and then made some changes to **index.html** and then built it again. So we see many files being cached during the second build.

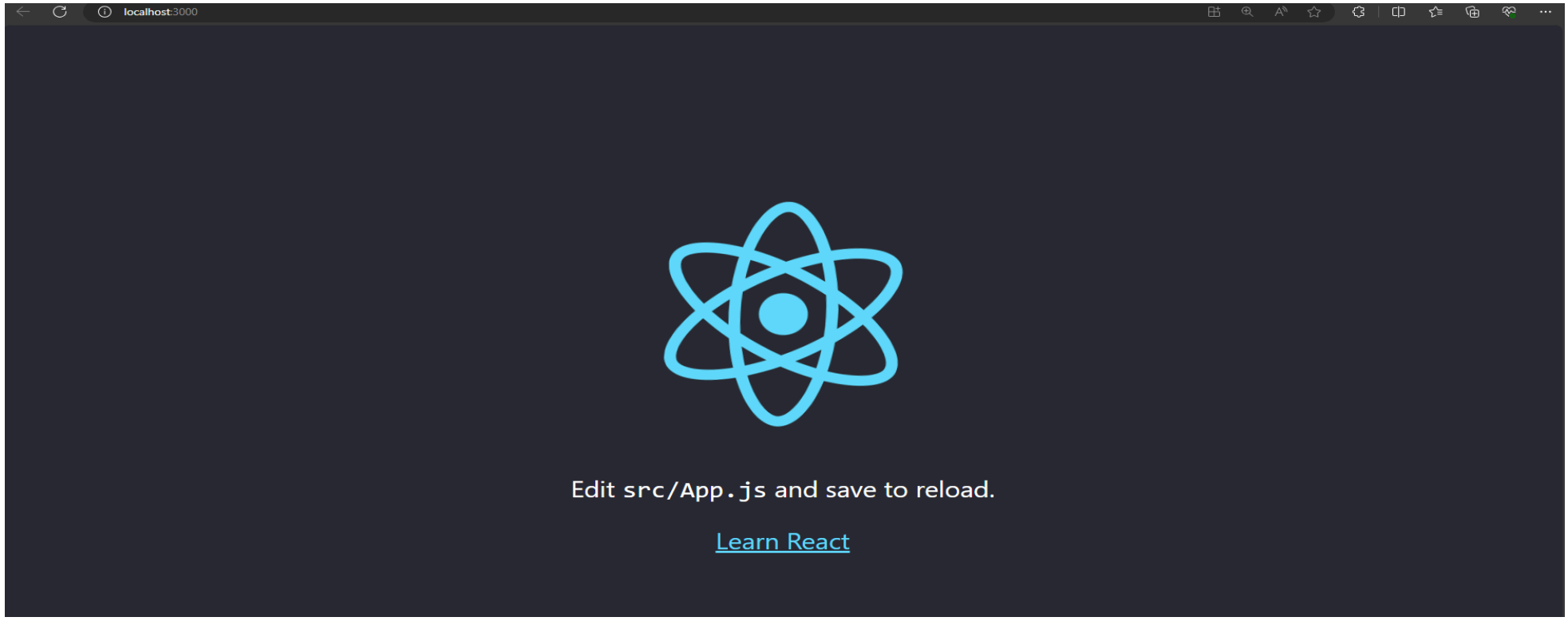


ii. Multi-stage builds

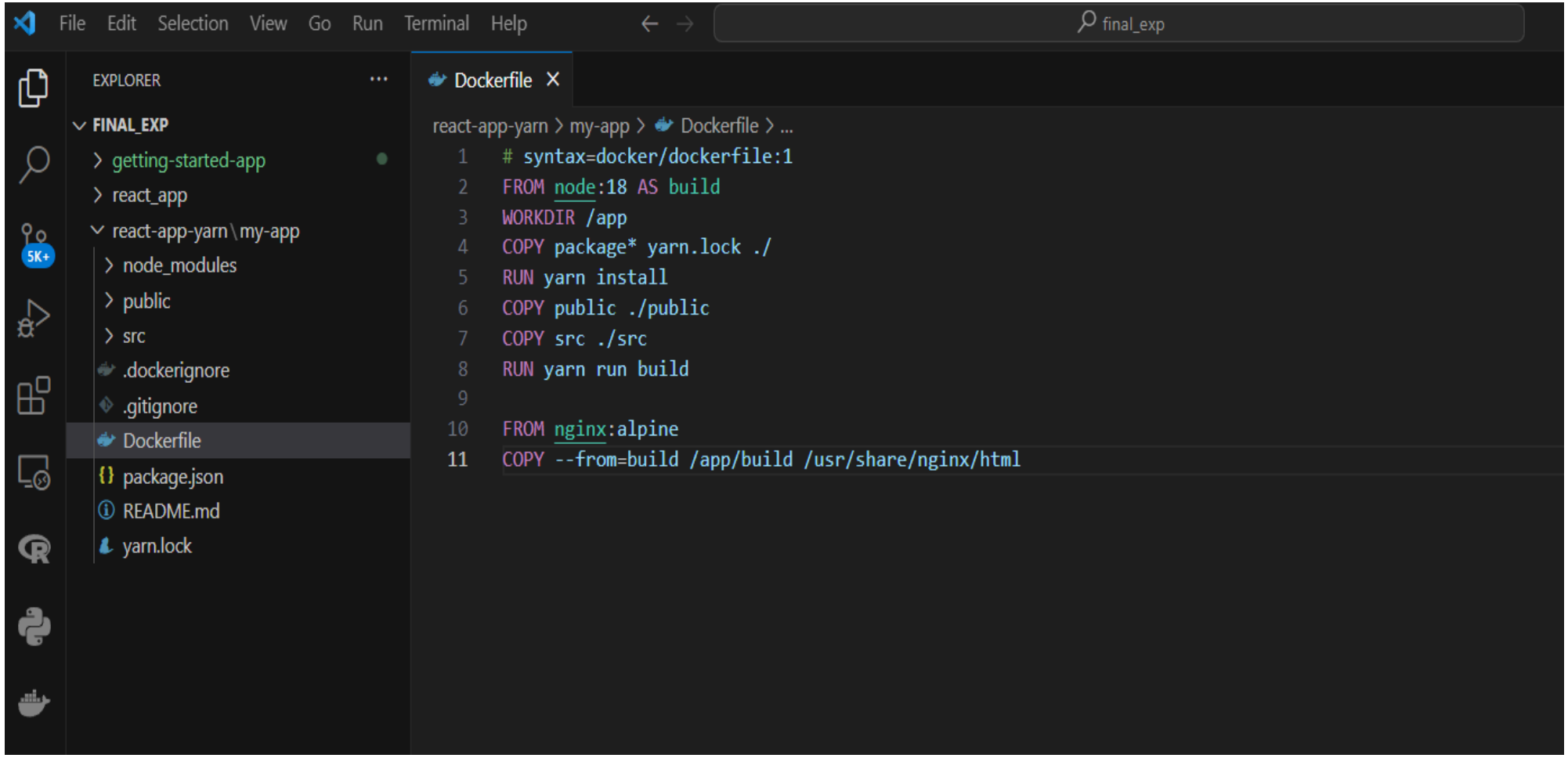
1. React example

a. Created a react app using command “**yarn create react-app my-app**” as this create a new react app called my-app and started it using “**npm start**” (ref:[Getting Started | Create React App (create-react-app.dev)](create-react-app.dev))



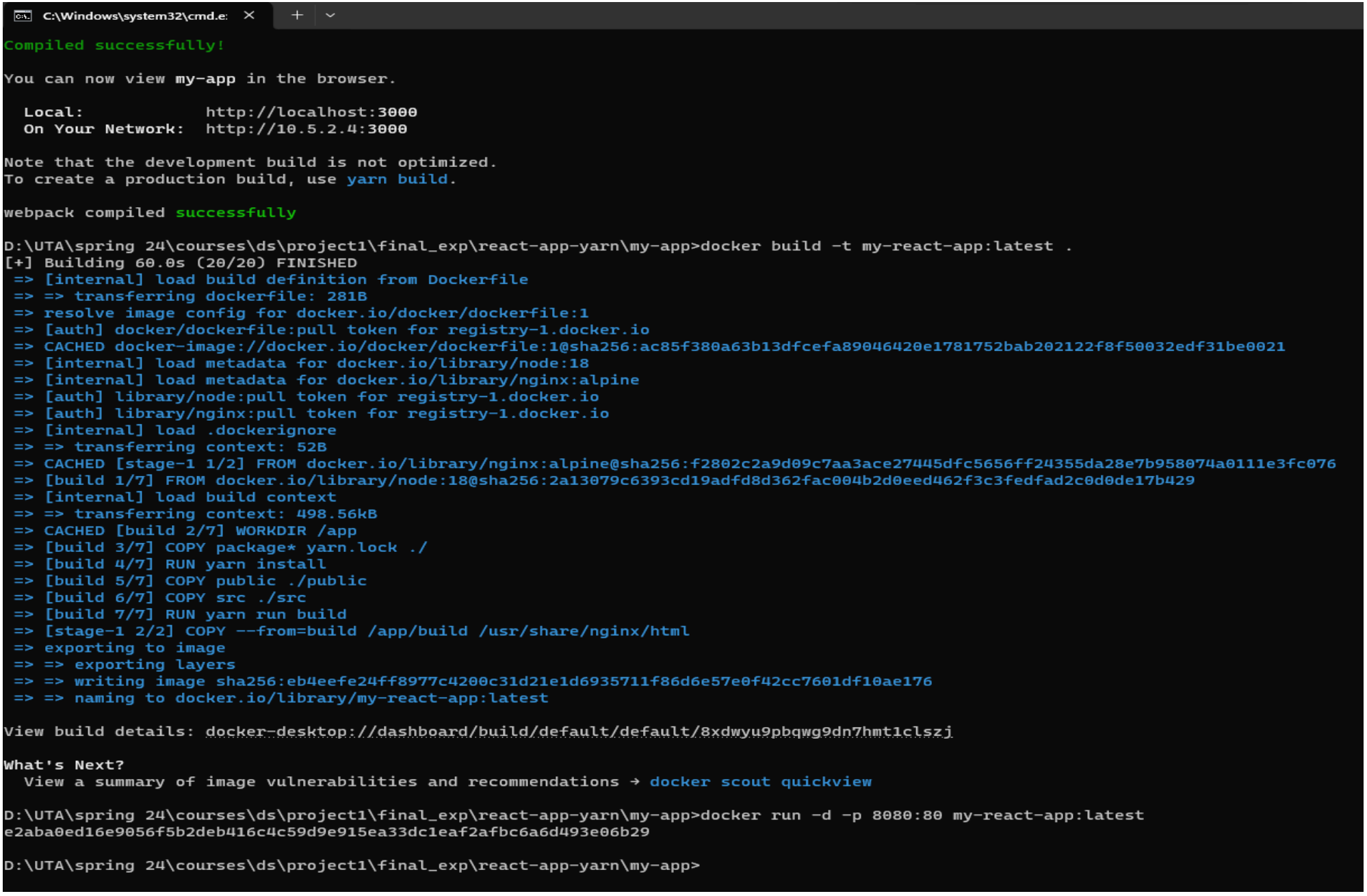


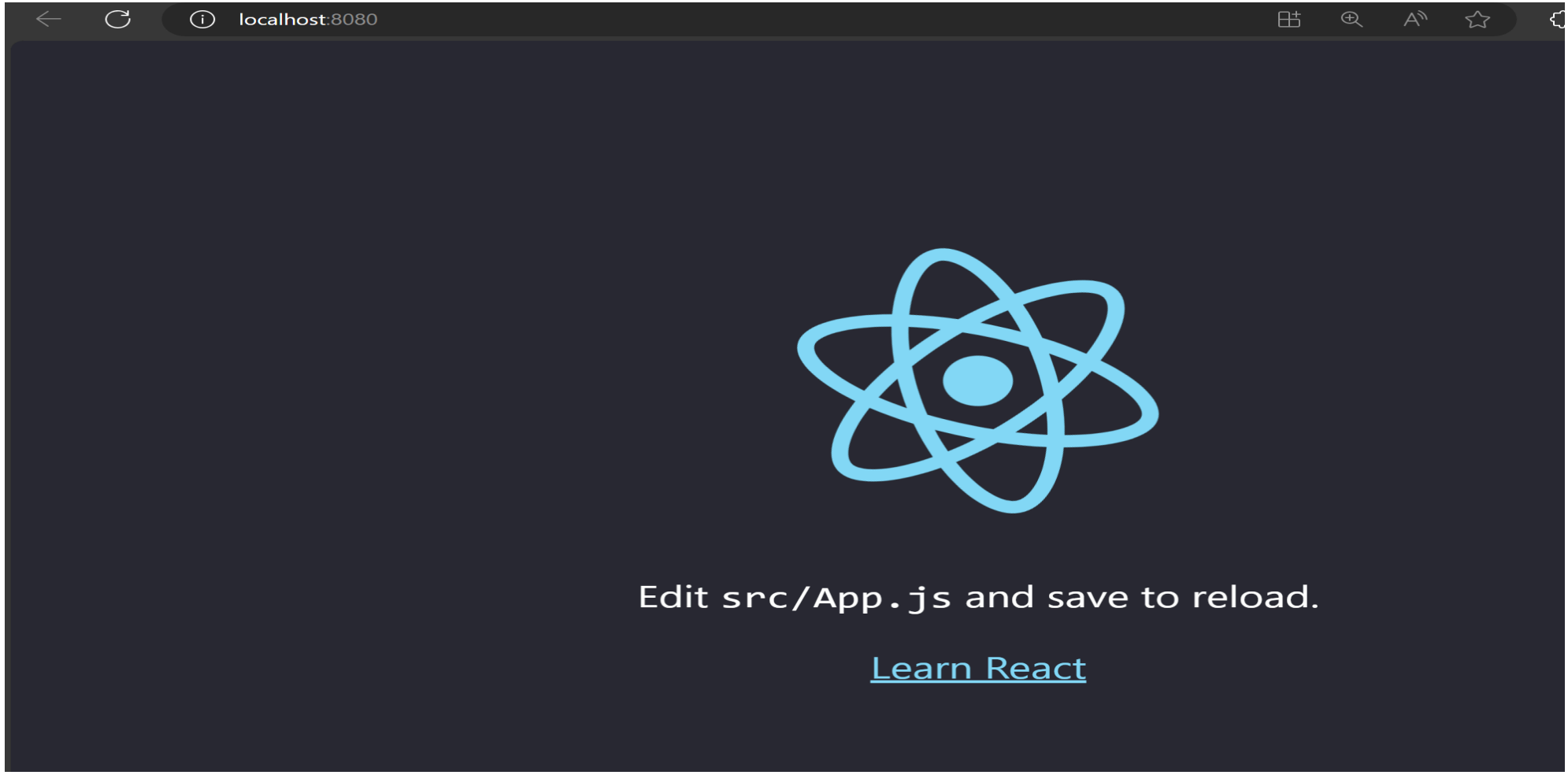
i. Created a docker file as below referring to the tutorial for react apps in docker for Multi-stage builds



ii. Created build, Ran the image container and the output is shown below.

1. As specified in Dockerfile above. Here we ship the static resources in a static nginx container as we are not doing any server side rendering.


C:\Windows\system32\cmd.e:
Compiled successfully!
You can now view my-app in the browser.
Local: http://localhost:3000
On Your Network: http://10.5.2.4:3000
Note that the development build is not optimized.
To create a production build, use yarn build.
webpack compiled successfully
D:\UTA\spring 24\courses\ds\project1\final_exp\react-app-yarn\my-app>docker build -t my-react-app:latest .
[+] Building 60.0s (20/20) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 281B
=> resolve image config for docker.io/docker/dockerfile:1
=> [auth] docker/dockerfile:pull token for registry-1.docker.io
=> CACHED docker-image://docker.io/docker/dockerfile:1@sha256:ac85f380a63b13dfcefa89046420e1781752bab202122f8f50032edf31be0021
=> [internal] load metadata for docker.io/library/node:18
=> [internal] load metadata for docker.io/library/nginx:alpine
=> [auth] library/node:pull token for registry-1.docker.io
=> [auth] library/nginx:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 52B
=> CACHED [stage-1 1/2] FROM docker.io/library/nginx:alpine@sha256:f2802c2a9d09c7aa3ace27445dfc5656ff24355da28e7b958074a0111e3fc076
=> [build 1/7] FROM docker.io/library/node:18@sha256:2a13079c6393cd19adfd8d362fac004b2d0eed462f3c3fedfad2c0d0de17b429
=> [internal] load build context
=> => transferring context: 498.56kB
=> CACHED [build 2/7] WORKDIR /app
=> [build 3/7] COPY package* yarn.lock ./
=> [build 4/7] RUN yarn install
=> [build 5/7] COPY public ./public
=> [build 6/7] COPY src ./src
=> [build 7/7] RUN yarn run build
=> [stage-1 2/2] COPY --from=build /app/build /usr/share/nginx/html
=> exporting to image
=> => exporting layers
=> => writing image sha256:eb4eefe24ff8977c4200c31d21e1d6935711f86d6e57e0f42cc7601df10ae176
=> => naming to docker.io/library/my-react-app:latest
View build details: docker-desktop://dashboard/build/default/default/8xdwyu9pbqwg9dn7hmt1clszj
What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview
D:\UTA\spring 24\courses\ds\project1\final_exp\react-app-yarn\my-app>docker run -d -p 8080:80 my-react-app:latest
e2aba0ed16e9056f5b2deb416c4c59d9e915ea33dc1eaf2afbc6a6d493e06b29
D:\UTA\spring 24\courses\ds\project1\final_exp\react-app-yarn\my-app>


localhost:8080
Edit src/App.js and save to reload.
Learn React

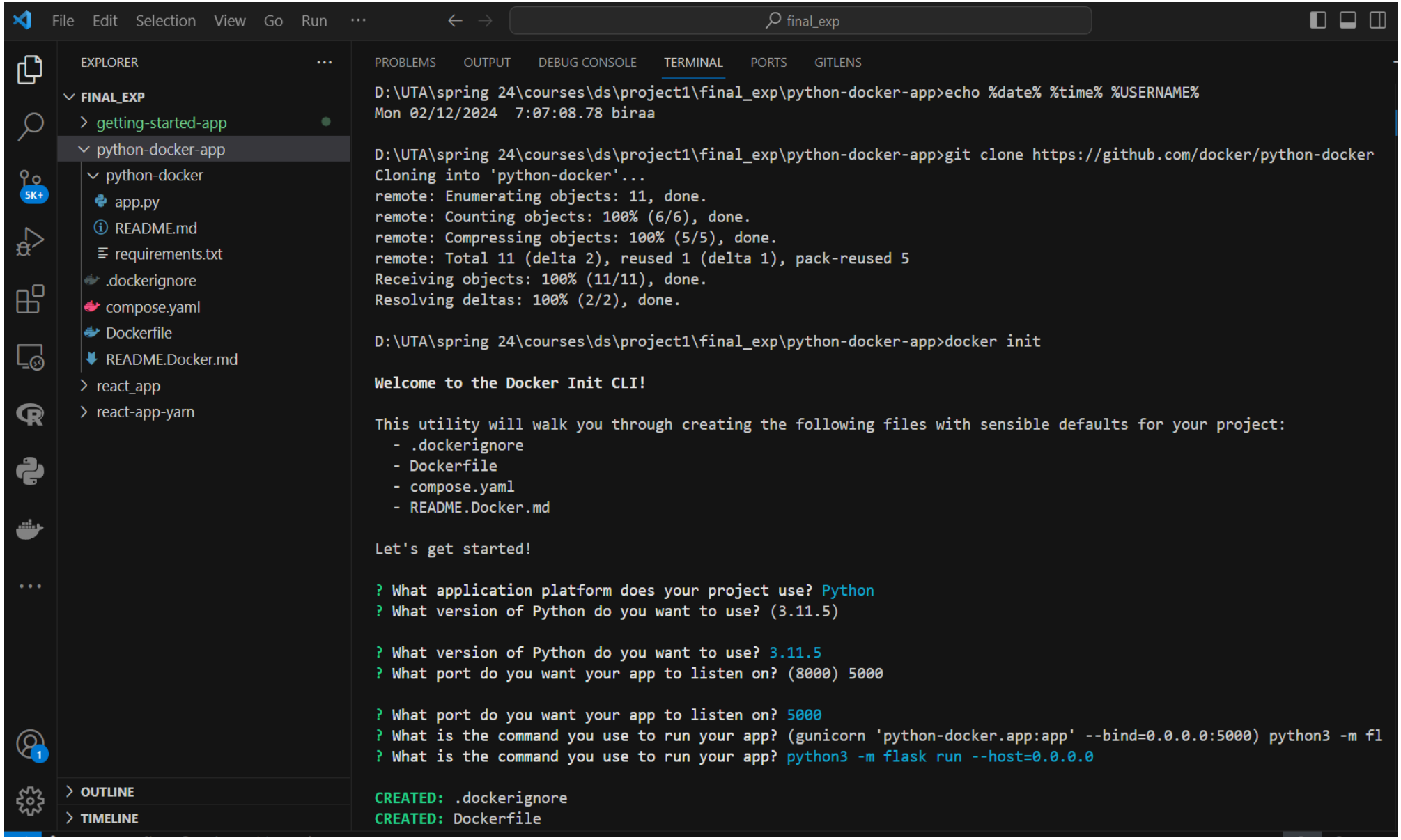
2. Python language-specific guide

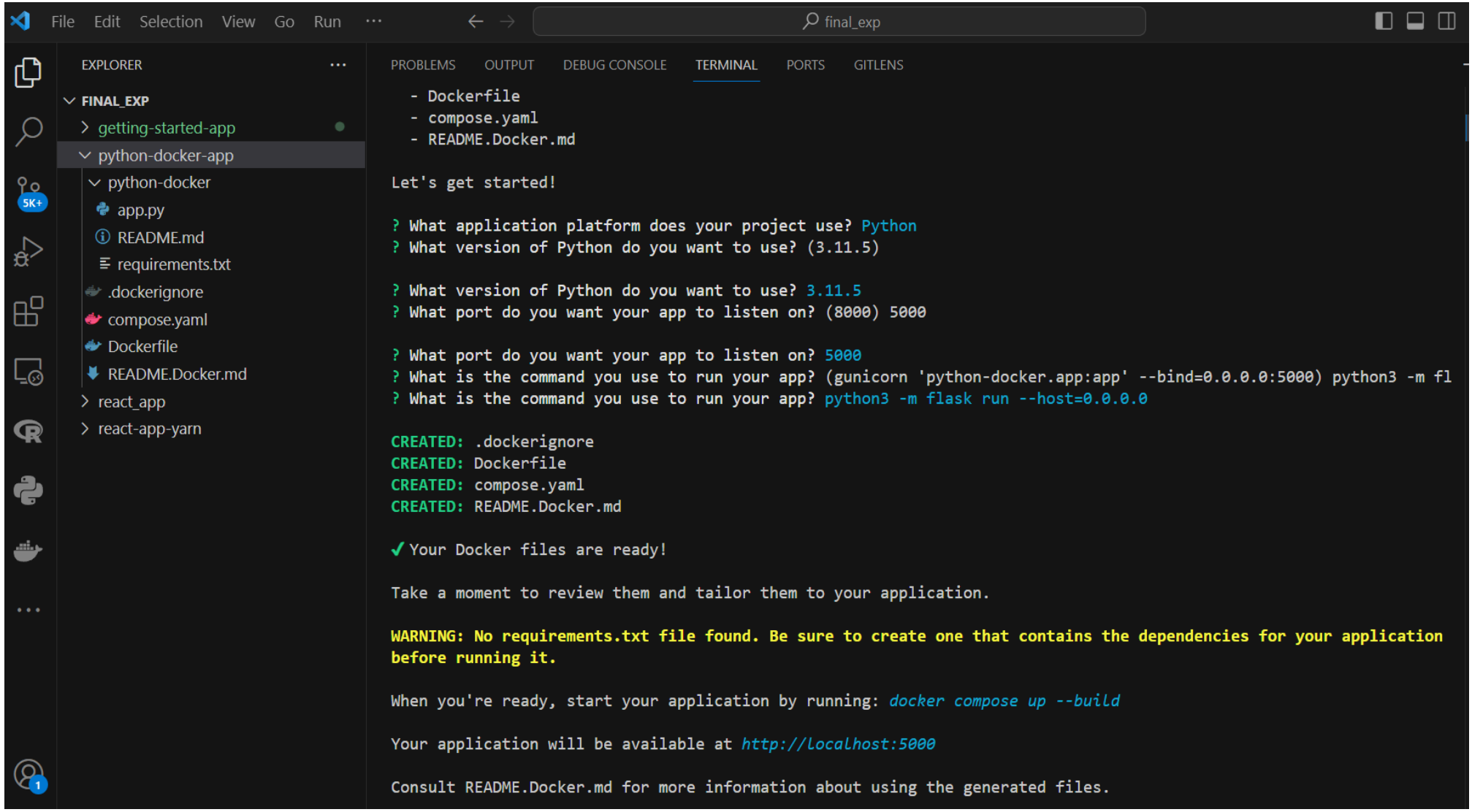
a. Desicription:

i. Here we are building images, running containers and developing an app using python and flask.

b. Containerize a Python application:

i. Cloned the same application from git using “git clone https://github.com/docker/python-docker” and Intialized docker application using “docker init”

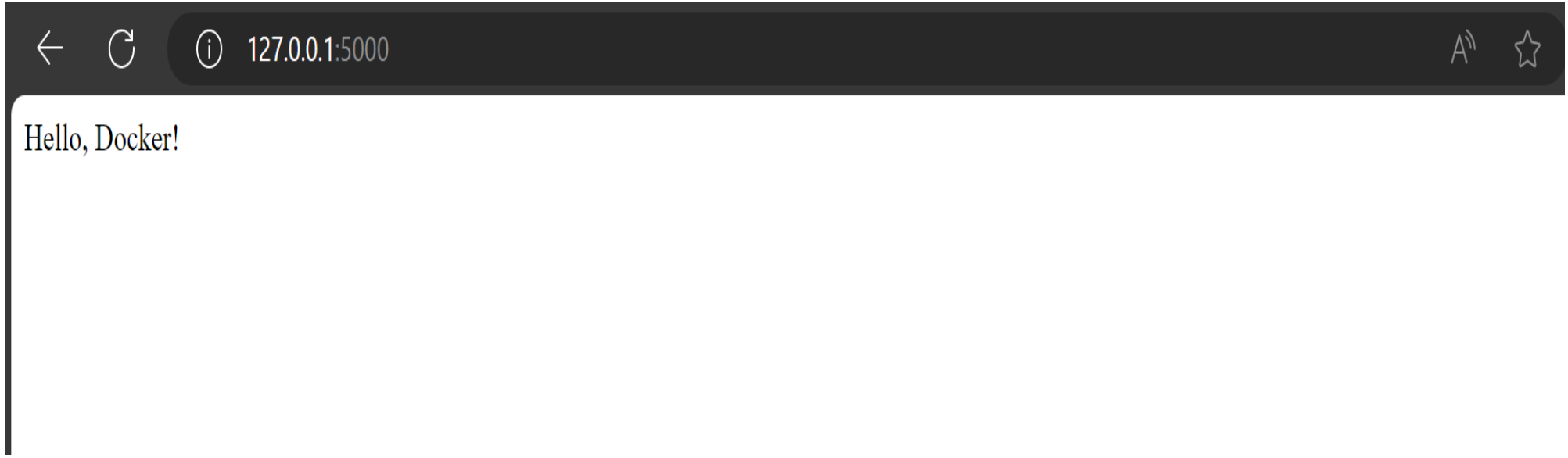

File Edit Selection View Go Run ···
final_exp
EXPLORER
FINAL_EXP
getting-started-app
python-docker-app
python-docker
app.py
README.md
requirements.txt
.dockerignore
compose.yaml
Dockerfile
README.Docker.md
react_app
react-app-yarn
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
D:\UTA\spring 24\courses\ds\project1\final_exp\python-docker-app>echo %date% %time% %USERNAME%
Mon 02/12/2024 7:07:08.78 biraa
D:\UTA\spring 24\courses\ds\project1\final_exp\python-docker-app>git clone https://github.com/docker/python-docker
Cloning into 'python-docker'...
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 11 (delta 2), reused 1 (delta 1), pack-reused 5
Receiving objects: 100% (11/11), done.
Resolving deltas: 100% (2/2), done.
D:\UTA\spring 24\courses\ds\project1\final_exp\python-docker-app>docker init
Welcome to the Docker Init CLI!
This utility will walk you through creating the following files with sensible defaults for your project:
- .dockerignore
- Dockerfile
- compose.yaml
- README.Docker.md
Let's get started!
? What application platform does your project use? Python
? What version of Python do you want to use? (3.11.5)
? What version of Python do you want to use? 3.11.5
? What port do you want your app to listen on? (8000) 5000
? What port do you want your app to listen on? 5000
? What is the command you use to run your app? (gunicorn 'python-docker.app:app' --bind=0.0.0.0:5000) python3 -m fl
? What is the command you use to run your app? python3 -m flask run --host=0.0.0.0
CREATED: .dockerignore
CREATED: Dockerfile

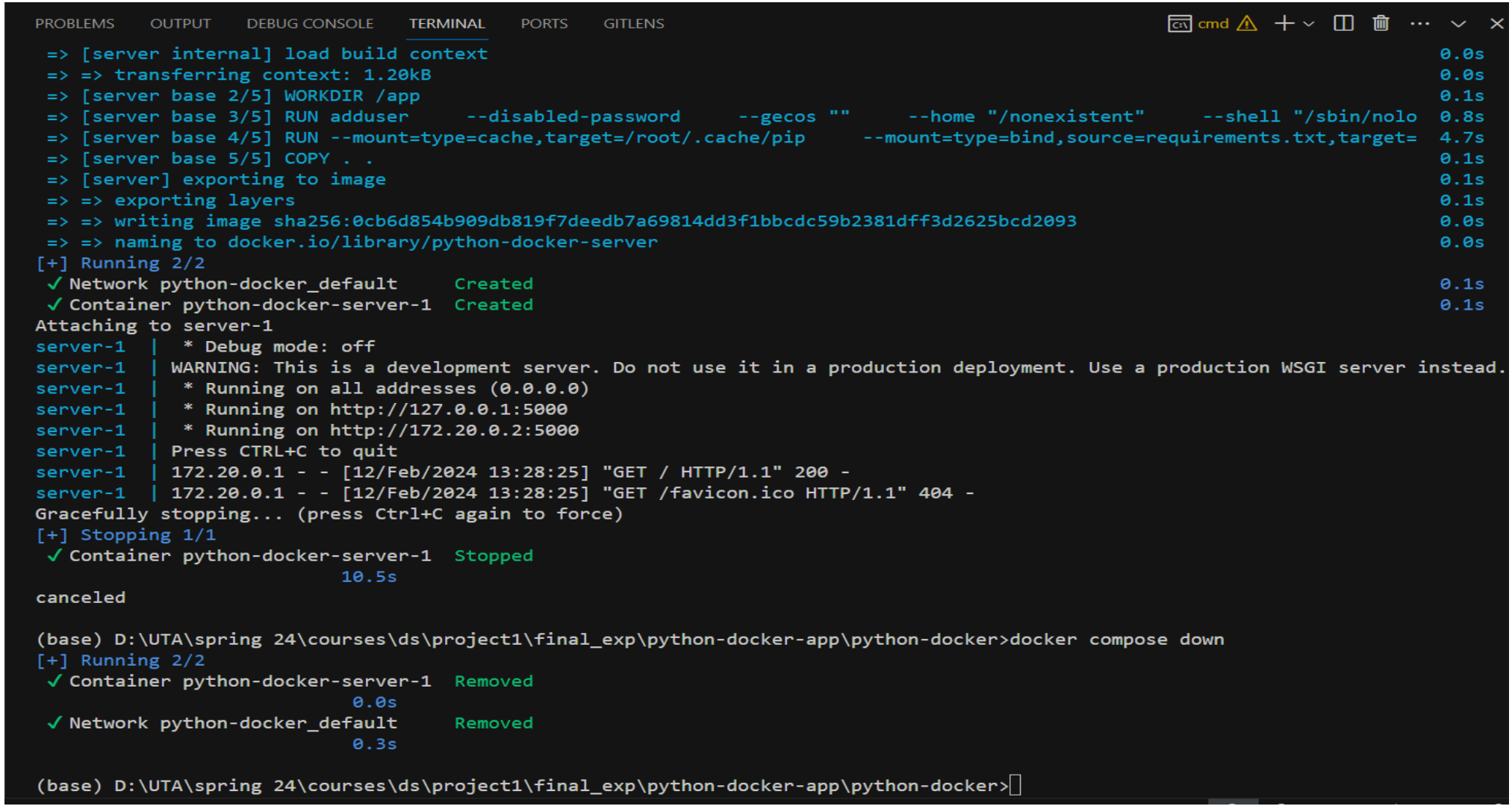

File Edit Selection View Go Run ···
final_exp
EXPLORER
FINAL_EXP
getting-started-app
python-docker-app
python-docker
app.py
README.md
requirements.txt
.dockerignore
compose.yaml
Dockerfile
README.Docker.md
react_app
react-app-yarn
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
- Dockerfile
- compose.yaml
- README.Docker.md
Let's get started!
? What application platform does your project use? Python
? What version of Python do you want to use? (3.11.5)
? What version of Python do you want to use? 3.11.5
? What port do you want your app to listen on? (8000) 5000
? What port do you want your app to listen on? 5000
? What is the command you use to run your app? (gunicorn 'python-docker.app:app' --bind=0.0.0.0:5000) python3 -m fl
? What is the command you use to run your app? python3 -m flask run --host=0.0.0.0
CREATED: .dockerignore
CREATED: Dockerfile
CREATED: compose.yaml
CREATED: README.Docker.md
✓ Your Docker files are ready!
Take a moment to review them and tailor them to your application.
WARNING: No requirements.txt file found. Be sure to create one that contains the dependencies for your application before running it.
When you're ready, start your application by running: docker compose up --build
Your application will be available at http://localhost:5000
Consult README.Docker.md for more information about using the generated files.

ii. Building the docker image and running it using docker compose

1. **docker compose up --build (**explained in part 8**)**
    a. Here --build flag triggers rebuilding images for services with changes in Dockerfiles
2. **docker compose down**(explained in part 8)

```
Mon 02/12/2024  7:25:37.91 biraa;

(base) D:\UTA\spring 24\courses\ds\project1\final_exp\python-docker-app\python-docker>docker compose up --build
[+] Building 0.0s (0/0)  docker:default
[+] Building 20.5s (14/14) FINISHED                                                                                 docker:default
 => [server internal] load build definition from Dockerfile                                                                  0.0s
 => => transferring dockerfile: 1.68kB                                                                                       0.0s
 => [server] resolve image config for docker.io/docker/dockerfile:1                                                          0.9s
 => [server auth] docker/dockerfile:pull token for registry-1.docker.io                                                      0.0s
 => CACHED [server] docker-image://docker.io/docker/dockerfile:1@sha256:ac85f380a63b13dfcefa89046420e1781752bab202122f8f50  0.0s
 => [server internal] load metadata for docker.io/library/python:3.11.4-slim                                                 0.9s
 => [server auth] library/python:pull token for registry-1.docker.io                                                         0.0s
 => [server internal] load .dockerignore                                                                                     0.0s
 => => transferring context: 667B                                                                                            0.0s
 => [server base 1/5] FROM docker.io/library/python:3.11.4-slim@sha256:17d62d681d9ecef20aae6c6605e9cf83b0ba3dc247013e2f43  12.5s
 => => sha256:17d62d681d9ecef20aae6c6605e9cf83b0ba3dc247013e2f43e1b5a045ad4901 1.65kB / 1.65kB                               0.0s
 => => sha256:0275089b5b654bb33931fc239a447db9fdd1628bc9d1482788754785d6d9e464 1.37kB / 1.37kB                               0.0s
 => => sha256:596e0d6b34dfaa7ed330941075bcd38b376b3eba8e5b63a1da38bf04fe08bdd3 6.92kB / 6.92kB                               0.0s
 => => sha256:52d2b7f179e32b4cbd579ee3c4958027988f9a8274850ab0c7c24661e3adaac5 29.12MB / 29.12MB                             3.6s
 => => sha256:2b8a9a2240c1224b34f6aafbc3310f9a3fe65bd6893050906d02e89fc8326aa9 3.50MB / 3.50MB                               0.3s
 => => sha256:051d6521462a7eb4ca0374e97701d6eec68eb51b118d3ef5d002798b498fb12e 17.86MB / 17.86MB                             3.1s
 => => sha256:fce84b1f897c621e9474bd4d5a49e2e22fa35e248e78e754010d34ec3d2d28cd 245B / 245B                                   0.6s
 => => sha256:46233543d8c2dc599bdb9d522180ca9e14cad4ac2017a5dc481660bfa4aa3ed9 3.38MB / 3.38MB                               2.2s
 => => extracting sha256:52d2b7f179e32b4cbd579ee3c4958027988f9a8274850ab0c7c24661e3adaac5                                    3.4s
 => => extracting sha256:2b8a9a2240c1224b34f6aafbc3310f9a3fe65bd6893050906d02e89fc8326aa9                                    0.4s
 => => extracting sha256:051d6521462a7eb4ca0374e97701d6eec68eb51b118d3ef5d002798b498fb12e                                    3.5s
 => => extracting sha256:fce84b1f897c621e9474bd4d5a49e2e22fa35e248e78e754010d34ec3d2d28cd                                    0.0s
 => => extracting sha256:46233543d8c2dc599bdb9d522180ca9e14cad4ac2017a5dc481660bfa4aa3ed9                                    1.1s
 => [server internal] load build context                                                                                     0.0s
 => => transferring context: 1.20kB                                                                                          0.0s
 => [server base 2/5] WORKDIR /app                                                                                           0.1s
 => [server base 3/5] RUN adduser     --disabled-password     --gecos ""     --home "/nonexistent"     --shell "/sbin/nolo  0.8s
 => [server base 4/5] RUN --mount=type=cache,target=/root/.cache/pip     --mount=type=bind,source=requirements.txt,target=  4.7s
 => [server base 5/5] COPY . .                                                                                               0.1s
 => [server] exporting to image                                                                                              0.1s
 => => exporting layers                                                                                                      0.1s
 => => writing image sha256:0cb6d854b909db819f7deedb7a69814dd3f1bbcdc59b2381dff3d2625bcd2093                                 0.0s
 => => naming to docker.io/library/python-docker-server                                                                      0.0s
[+] Running 2/2
 ✓ Network python-docker_default     Created                                                                                 0.1s
 ✓ Container python-docker-server-1  Created                                                                                 0.1s
Attaching to server-1
server-1  |  * Debug mode: off
```

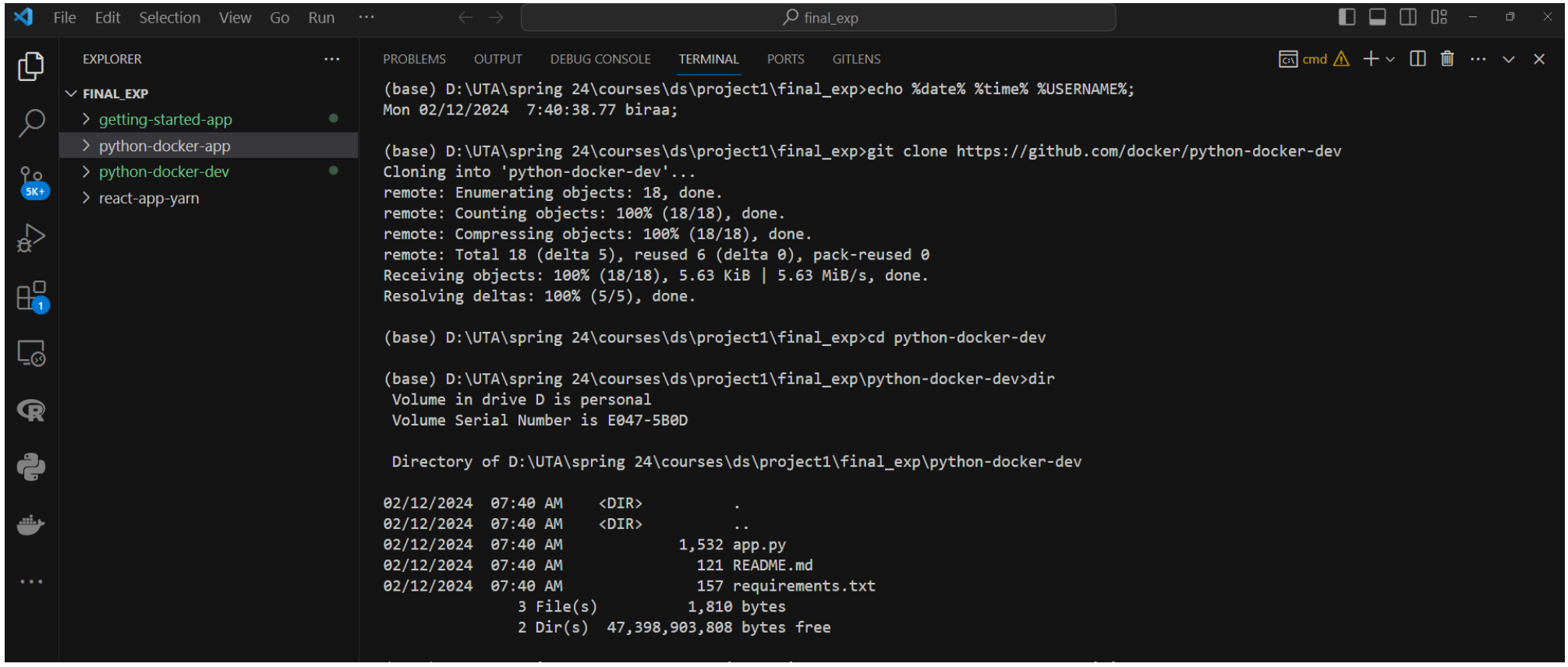

127.0.0.1:5000
Hello, Docker!


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
GITLENS
cmd
=> [server internal] load build context 0.0s
=> => transferring context: 1.20kB 0.0s
=> [server base 2/5] WORKDIR /app 0.1s
=> [server base 3/5] RUN adduser --disabled-password --gecos "" --home "/nonexistent" --shell "/sbin/nolo 0.8s
=> [server base 4/5] RUN --mount=type=cache,target=/root/.cache/pip --mount=type=bind,source=requirements.txt,target= 4.7s
=> [server base 5/5] COPY . . 0.1s
=> [server] exporting to image 0.1s
=> => exporting layers 0.1s
=> => writing image sha256:0cb6d854b909db819f7deedb7a69814dd3f1bbcdc59b2381dff3d2625bcd2093 0.0s
=> => naming to docker.io/library/python-docker-server 0.0s
[+] Running 2/2
✓ Network python-docker_default Created 0.1s
✓ Container python-docker-server-1 Created 0.1s
Attaching to server-1
server-1 | * Debug mode: off
server-1 | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
server-1 | * Running on all addresses (0.0.0.0)
server-1 | * Running on http://127.0.0.1:5000
server-1 | * Running on http://172.20.0.2:5000
server-1 | Press CTRL+C to quit
server-1 | 172.20.0.1 - - [12/Feb/2024 13:28:25] "GET / HTTP/1.1" 200 -
server-1 | 172.20.0.1 - - [12/Feb/2024 13:28:25] "GET /favicon.ico HTTP/1.1" 404 -
Gracefully stopping... (press Ctrl+C again to force)
[+] Stopping 1/1
✓ Container python-docker-server-1 Stopped 10.5s
canceled
(base) D:\UTA\spring 24\courses\ds\project1\final_exp\python-docker-app\python-docker>docker compose down
[+] Running 2/2
✓ Container python-docker-server-1 Removed 0.0s
✓ Network python-docker_default Removed 0.3s
(base) D:\UTA\spring 24\courses\ds\project1\final_exp\python-docker-app\python-docker>

c. Develop your App

i. Use containers for Python development

1. Cloned the sample application from “git clone https://github.com/docker/python-docker-dev”
2. Intialized the docker files with “**docker init**”. This command Initialize a project with the files necessary to run the project in a container.



```
(base) D:\UTA\spring 24\courses\ds\project1\final_exp\python-docker-dev>docker init

Welcome to the Docker Init CLI!

This utility will walk you through creating the following files with sensible defaults for your project:
  - .dockerignore
  - Dockerfile
  - compose.yaml
  - README.Docker.md

Let's get started!

? What application platform does your project use? Python
? What version of Python do you want to use? (3.11.5) 3.11.4

? What version of Python do you want to use? 3.11.4
? What port do you want your app to listen on? (8000) 5000

? What port do you want your app to listen on? 5000
? What is the command you use to run your app? (gunicorn 'app:app' --bind=0.0.0.0:5000) python3 -m flask run --host=0.0.0.0

? What is the command you use to run your app? python3 -m flask run --host=0.0.0.0

CREATED: .dockerignore
CREATED: Dockerfile
CREATED: compose.yaml
CREATED: README.Docker.md

✔ Your Docker files are ready!

Take a moment to review them and tailor them to your application.

When you're ready, start your application by running: docker compose up --build

Your application will be available at http://localhost:5000

Consult README.Docker.md for more information about using the generated files.
```

ii. Add a local database and persist data

1. Updating the **compose.yaml** file to have all the database instructions
2. Before running the application we created db folder and a file called **password.txt** to store our password which is "**mysecretpaswword**"

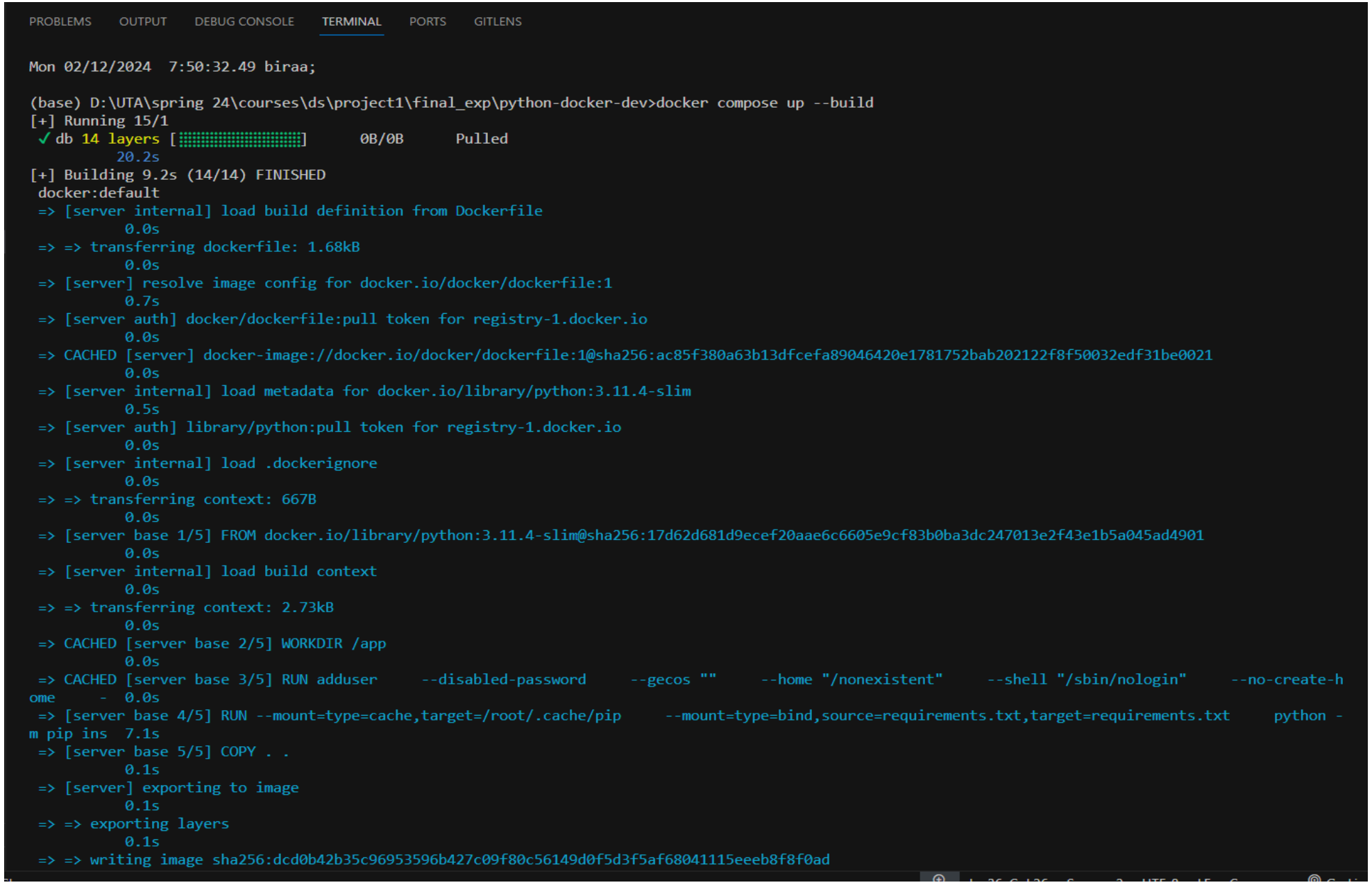
compose.yaml U | password.txt U

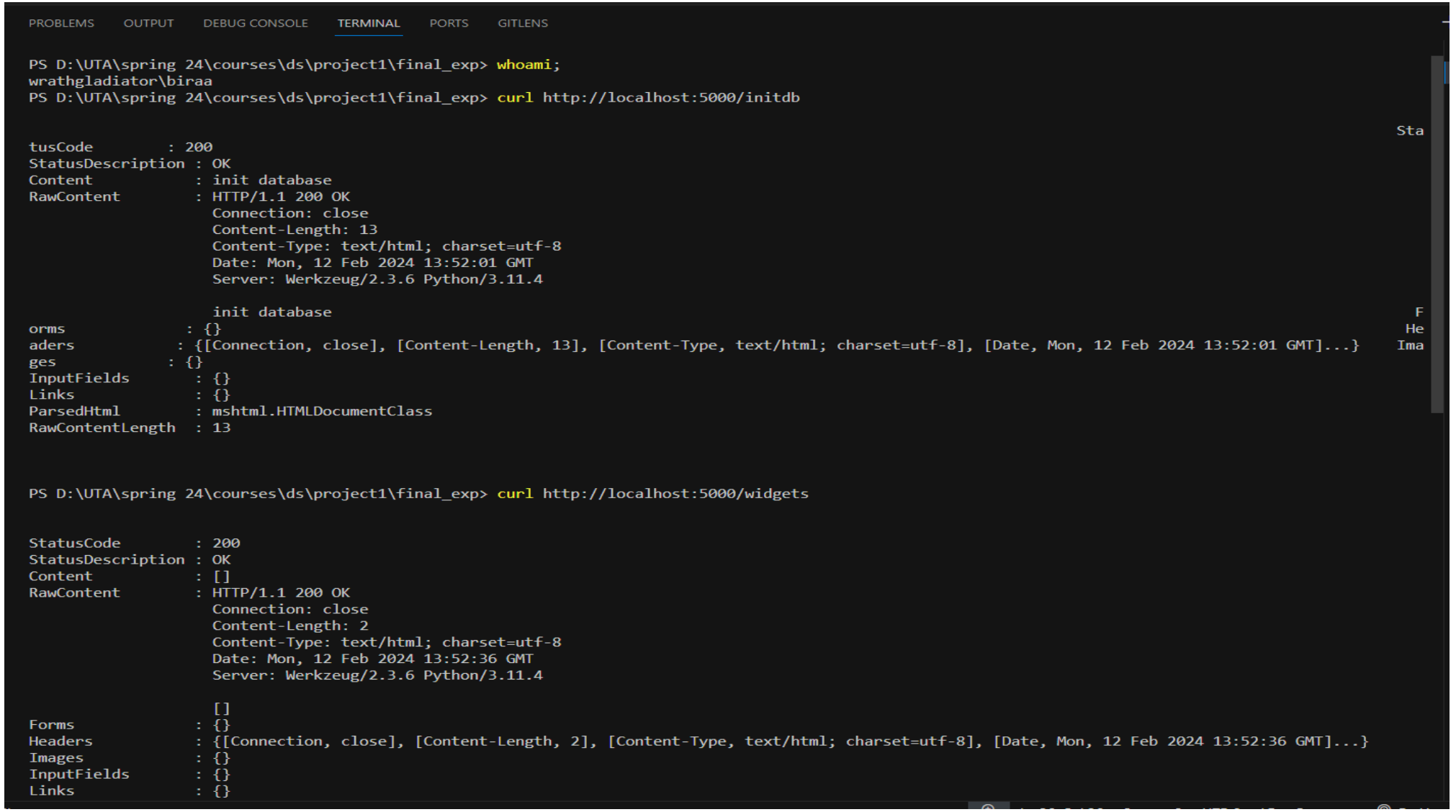
python-docker-dev > compose.yaml > {} secrets > {} db-password > file

docker-compose.yml - The Compose specification establishes a standard for the definition of multi-container platform-agnostic applications (compose-spec.json)

```yaml
services:
  server:
    build:
      context: .
    ports:
      - 5000:5000
    environment:
      - POSTGRES_PASSWORD_FILE=/run/secrets/db-password
    depends_on:
      db:
        condition: service_healthy
    secrets:
      - db-password
  db:
    image: postgres
    restart: always
    user: postgres
    secrets:
      - db-password
    volumes:
      - db-data:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB=example
      - POSTGRES_PASSWORD_FILE=/run/secrets/db-password
    expose:
      - 5432
    healthcheck:
      test: [ "CMD", "pg_isready" ]
      interval: 10s
      timeout: 5s
      retries: 5
volumes:
  db-data:
secrets:
  db-password:
    file: db/password.txt
```

3. Aftret that we build and run the docker using “**docker compose up –build**” and try out the below commands and get empty response as our database is empty.
    a. curl http://localhost:5000/initdb
    b. curl http://localhost:5000/widgets
    c. Curl is is a command line tool that enables data exchange between a device and a server through a terminal.

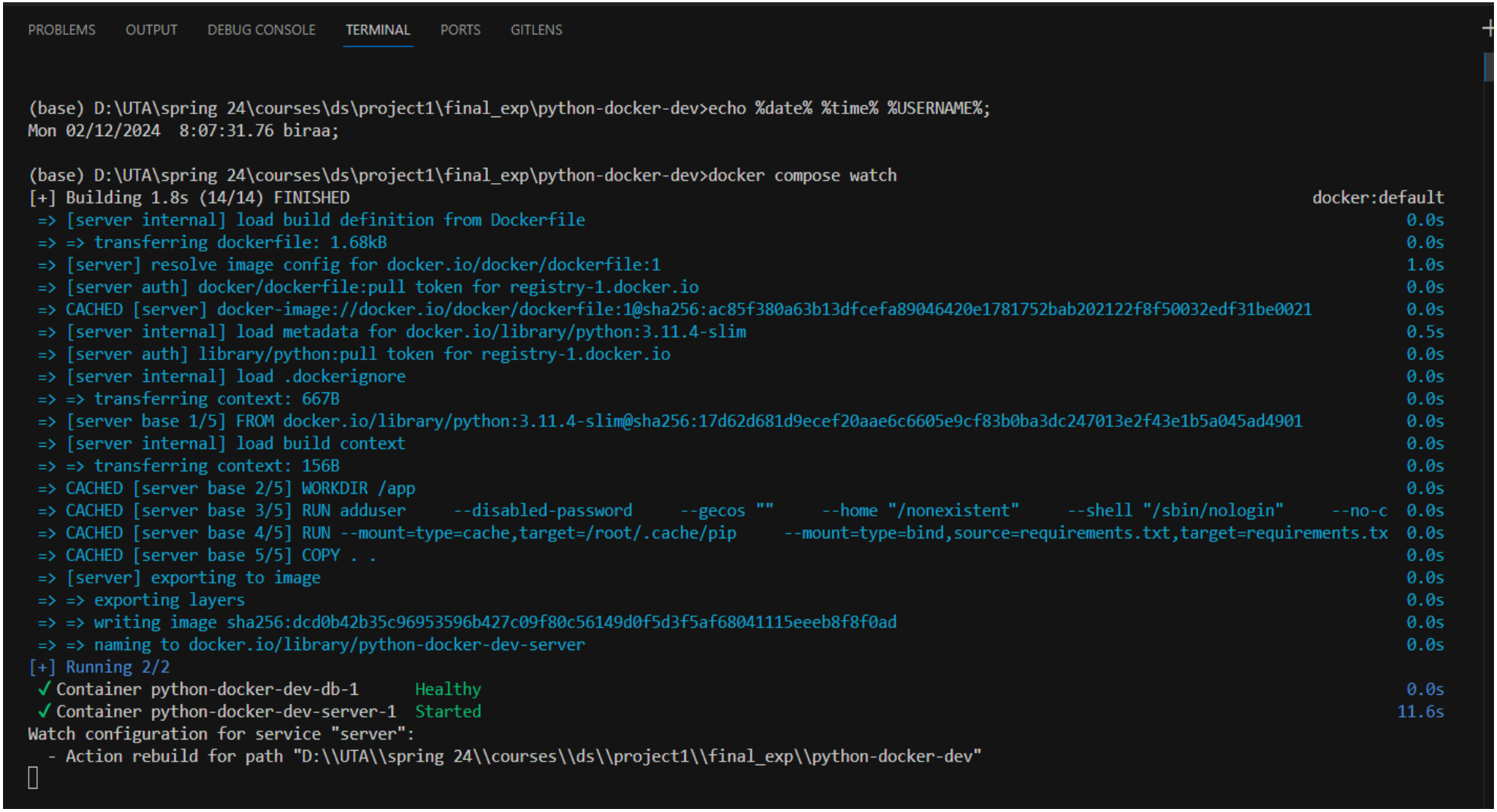

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
GITLENS
Mon 02/12/2024 7:50:32.49 biraa;
(base) D:\UTA\spring 24\courses\ds\project1\final_exp\python-docker-dev>docker compose up --build
[+] Running 15/1
✓ db 14 layers [] 0B/0B Pulled
20.2s
[+] Building 9.2s (14/14) FINISHED
docker:default
=> [server internal] load build definition from Dockerfile
0.0s
=> => transferring dockerfile: 1.68kB
0.0s
=> [server] resolve image config for docker.io/docker/dockerfile:1
0.7s
=> [server auth] docker/dockerfile:pull token for registry-1.docker.io
0.0s
=> CACHED [server] docker-image://docker.io/docker/dockerfile:1@sha256:ac85f380a63b13dfcefa89046420e1781752bab202122f8f50032edf31be0021
0.0s
=> [server internal] load metadata for docker.io/library/python:3.11.4-slim
0.5s
=> [server auth] library/python:pull token for registry-1.docker.io
0.0s
=> [server internal] load .dockerignore
0.0s
=> => transferring context: 667B
0.0s
=> [server base 1/5] FROM docker.io/library/python:3.11.4-slim@sha256:17d62d681d9ecef20aae6c6605e9cf83b0ba3dc247013e2f43e1b5a045ad4901
0.0s
=> [server internal] load build context
0.0s
=> => transferring context: 2.73kB
0.0s
=> CACHED [server base 2/5] WORKDIR /app
0.0s
=> CACHED [server base 3/5] RUN adduser --disabled-password --gecos "" --home "/nonexistent" --shell "/sbin/nologin" --no-create-home - 0.0s
=> [server base 4/5] RUN --mount=type=cache,target=/root/.cache/pip --mount=type=bind,source=requirements.txt,target=requirements.txt python -m pip ins 7.1s
=> [server base 5/5] COPY . .
0.1s
=> [server] exporting to image
0.1s
=> => exporting layers
0.1s
=> => writing image sha256:dcd0b42b35c96953596b427c09f80c56149d0f5d3f5af68041115eeeb8f8f0ad



4. We then stop the the application using **ctrl+c**

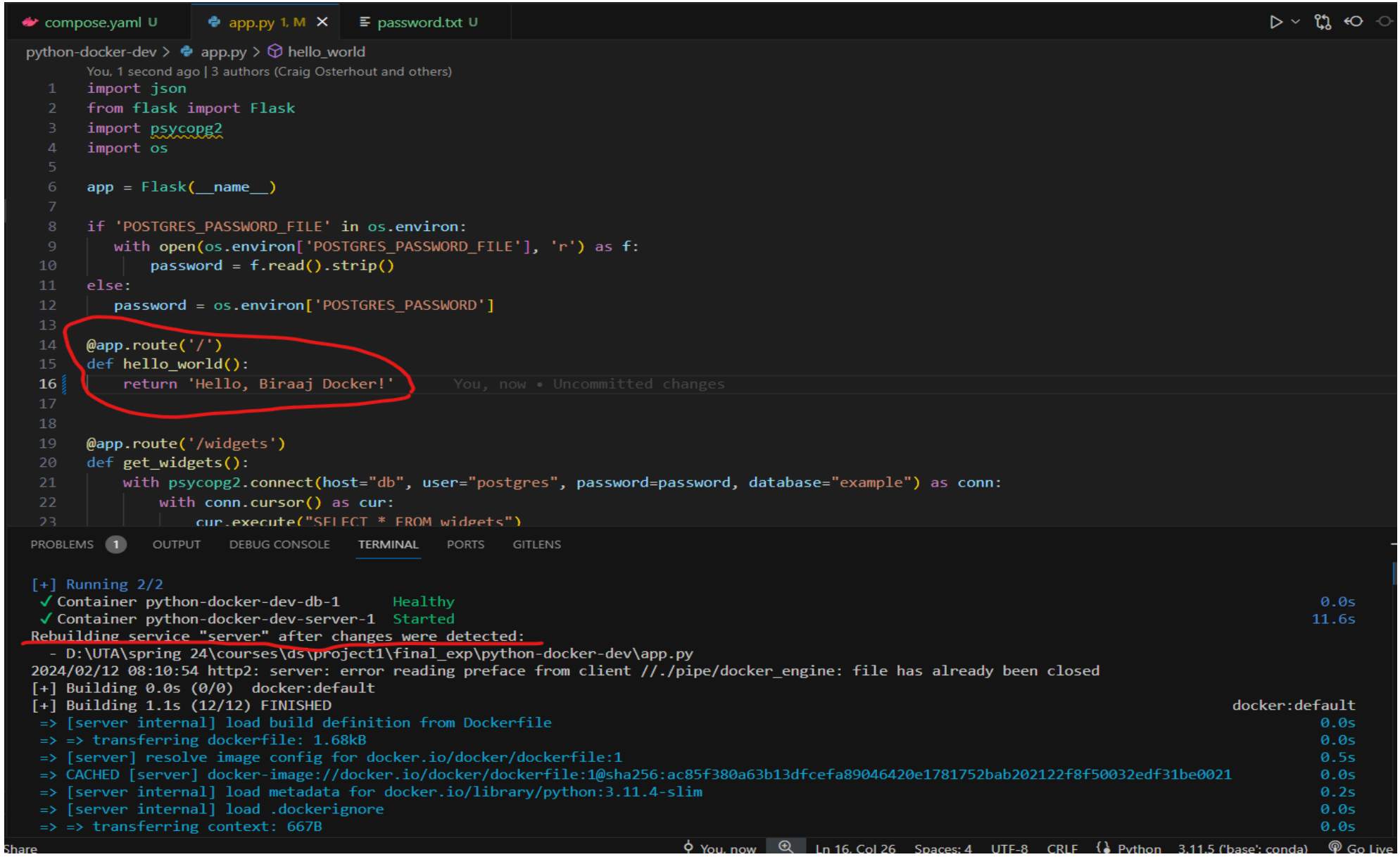
iii. Automatically update services

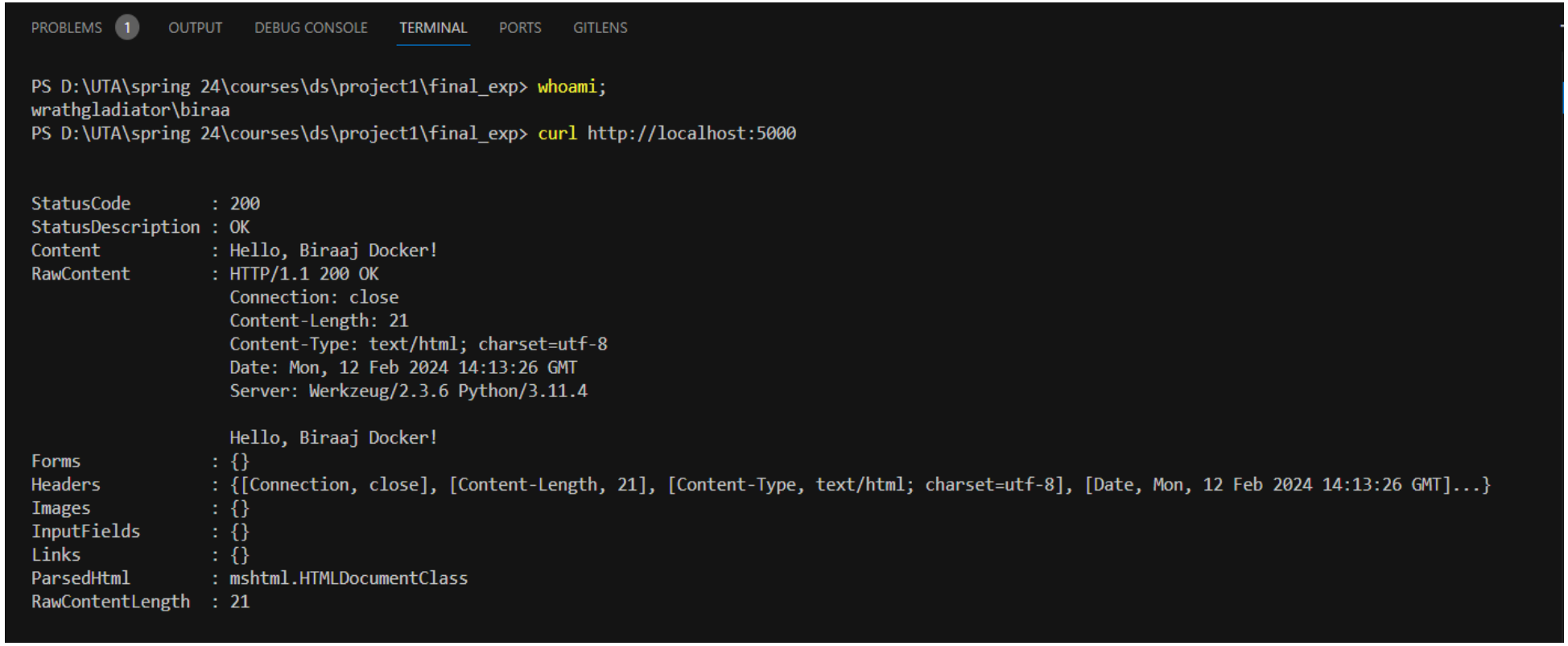
1. Used compose watch to automatically update running compose services as we edit and save our code. To enable that we modified **compose.yaml** and then run "**docker compose watch**" to watch for changes.



```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   GITLENS

PS D:\UTA\spring 24\courses\ds\project1\final_exp> whoami;
wrathgladiator\biraa
PS D:\UTA\spring 24\courses\ds\project1\final_exp> curl http://localhost:5000


StatusCode        : 200
StatusDescription : OK
Content           : Hello, Docker!
RawContent        : HTTP/1.1 200 OK
                    Connection: close
                    Content-Length: 14
                    Content-Type: text/html; charset=utf-8
                    Date: Mon, 12 Feb 2024 14:09:10 GMT
                    Server: Werkzeug/2.3.6 Python/3.11.4

                    Hello, Docker!
Forms             : {}
Headers           : {[Connection, close], [Content-Length, 14], [Content-Type, text/html; charset=utf-8], [Date, Mon, 12 Feb 2024 14:09:10 GMT]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 14
```

2. After changing the **app.py** we see that the docker compose builds the image again because docker watch is running and we see the modified response when we hit the "**curl** http://localhost:5000"


compose.yaml U
app.py 1, M
password.txt U
python-docker-dev > app.py > hello_world
You, 1 second ago | 3 authors (Craig Osterhout and others)
import json
from flask import Flask
import psycopg2
import os
app = Flask(__name__)
if 'POSTGRES_PASSWORD_FILE' in os.environ:
with open(os.environ['POSTGRES_PASSWORD_FILE'], 'r') as f:
password = f.read().strip()
else:
password = os.environ['POSTGRES_PASSWORD']
@app.route('/')
def hello_world():
return 'Hello, Biraaj Docker!'
You, now • Uncommitted changes
@app.route('/widgets')
def get_widgets():
with psycopg2.connect(host="db", user="postgres", password=password, database="example") as conn:
with conn.cursor() as cur:
cur.execute("SELECT * FROM widgets")
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
GITLENS
[+] Running 2/2
✓ Container python-docker-dev-db-1 Healthy 0.0s
✓ Container python-docker-dev-server-1 Started 11.6s
Rebuilding service "server" after changes were detected:
- D:\UTA\spring 24\courses\ds\project1\final_exp\python-docker-dev\app.py
2024/02/12 08:10:54 http2: server: error reading preface from client //./pipe/docker_engine: file has already been closed
[+] Building 0.0s (0/0) docker:default
[+] Building 1.1s (12/12) FINISHED docker:default
=> [server internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 1.68kB 0.0s
=> [server] resolve image config for docker.io/docker/dockerfile:1 0.5s
=> CACHED [server] docker-image://docker.io/docker/dockerfile:1@sha256:ac85f380a63b13dfcefa89046420e1781752bab202122f8f50032edf31be0021 0.0s
=> [server internal] load metadata for docker.io/library/python:3.11.4-slim 0.2s
=> [server internal] load .dockerignore 0.0s
=> => transferring context: 667B 0.0s
Share
You, now
Ln 16, Col 26
Spaces: 4
UTF-8
CRLF
Python
3.11.5 ('base': conda)
Go Live


PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
GITLENS
PS D:\UTA\spring 24\courses\ds\project1\final_exp> whoami;
wrathgladiator\biraa
PS D:\UTA\spring 24\courses\ds\project1\final_exp> curl http://localhost:5000
StatusCode : 200
StatusDescription : OK
Content : Hello, Biraaj Docker!
RawContent : HTTP/1.1 200 OK
Connection: close
Content-Length: 21
Content-Type: text/html; charset=utf-8
Date: Mon, 12 Feb 2024 14:13:26 GMT
Server: Werkzeug/2.3.6 Python/3.11.4
Hello, Biraaj Docker!
Forms : {}
Headers : {[Connection, close], [Content-Length, 21], [Content-Type, text/html; charset=utf-8], [Date, Mon, 12 Feb 2024 14:13:26 GMT]...}
Images : {}
InputFields : {}
Links : {}
ParsedHtml : mshtml.HTMLDocumentClass
RawContentLength : 21

References:

1. https://docs.docker.com/get-started/overview/
2. https://docs.docker.com/language/python/
3. Git Guides - git clone (github.com)
4. https://docs.docker.com/engine/reference/builder/
5. https://create-react-app.dev/docs/getting-started
6. https://docs.docker.com/engine/reference/commandline
7. https://blog.hubspot.com/website/curl-command