Creating Docker Images for LMS Application.

Step 1: Configuring Postgres Database

Created a Postgres container directly without specifying the password so the container was unable to start so we can check / trouble shoot the container using the command :

```
azureuser@azure:~/lms$ docker container logs t1
Error: Database is uninitialized and superuser password is not specified.
       You must specify POSTGRES_PASSWORD to a non-empty value for the
       superuser. For example, "-e POSTGRES_PASSWORD=password" on "docker run".

       You may also use "POSTGRES_HOST_AUTH_METHOD=trust" to allow all
       connections without a password. This is *not* recommended.

       See PostgreSQL documentation about "trust":
       https://www.postgresql.org/docs/current/auth-trust.html
azureuser@azure:~/lms$
```

The actual command for setting up postgres database container is:

Docker container run -dt –name **“container_name”** -e POSTGRES_PASSWORD=**”PASSWORD”** postgres

```
azureuser@azure: ~/lms
azureuser@azure:~/lms$ docker container run -dt --name lms-db -e POSTGRES_PASSWORD = lms123 postgres
docker: invalid reference format.
See 'docker run --help'.
azureuser@azure:~/lms$ docker container run -dt --name lms-db -e POSTGRES_PASSWORD=lms123 postgres
9ce6390d8d8ae5ad3cbf7def6f9109897160d3daad13a2fcc47ee97a22d4bfec
azureuser@azure:~/lms$ docker container ls
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS      NAMES
9ce6390d8d8a   postgres   "docker-entrypoint.s…"   28 seconds ago   Up 28 seconds   5432/tcp   lms-db
azureuser@azure:~/lms$
```

Step 2: Building Docker Image for LMS BACKEND:

To build the we need to install node js

Command to create a node container is :
docker container run -dt - - name n1 node:16

```
azureuser@azure: ~
azureuser@azure:~$ docker container run -dt - - name n1 node:16
docker: invalid reference format.
See 'docker run --help'.
azureuser@azure:~$ docker container run -dt -- name n1 node:16
Unable to find image 'name:latest' locally
docker: Error response from daemon: pull access denied for name, repository does not exist or may requi
re 'docker login': denied: requested access to the resource is denied.
See 'docker run --help'.
azureuser@azure:~$ docker container run -dt --name n1 node:16
Unable to find image 'node:16' locally
16: Pulling from library/node
311da6c465ea: Pull complete
7e9bf114588c: Pull complete
ffd9397e94b7: Pull complete
513d77925604: Pull complete
ae3b95bbaa61: Pull complete
0e421f66aff4: Pull complete
ca266fd61921: Pull complete
ee7d78be1eb9: Pull complete
Digest: sha256:f77a1aef2da8d83e45ec990f45df50f1a286c5fe8bbfb8c6e4246c6389705c0b
Status: Downloaded newer image for node:16
06b5d9bab3bc38b06fc6737c5a353682616c41160f528a4a00bef77d5a4b03f1
azureuser@azure:~$ docker container exec n1 node -v
v16.20.2
azureuser@azure:~$ docker container exec n1 npm -v
8.19.4
azureuser@azure:~$
```

Once the node js and npm is installed then need to connect the backend “api” to database by providing credentials of database into .env file.

```
azureuser@azure: ~/lms/api
MODE=production
PORT=8080
DATABASE_URL=postgresql://postgres:lms123@172.17.0.2:5432/postgres
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
".env" 4L, 94C                                    1,1           All
```

Now we can create the docker file and write the commands to create a separate directry

1. mkdir to create a directry
2. wkdir to make it as present work directry
3. need to copy all the files from backend directry such as file present in api into the backend directry in the container
4. npm install command is used to install all the node modules which are present package.json file
5. npm prisma db push : is used to insert the tables data into the database
6. npm run build used to build the "**Build"** folder in backend directry
7. we can expose port 8080 or any port so that the application can be started in that port command for exposing the port is **Expose "port_number"**
8. command for building the file is build/index.js to write the same command in Docker file → **CMD ["node","build/index.js"]**

so the final Docker file is below:

```
azureuser@azure: ~/lms/api
FROM node:16
RUN mkdir /backend
WORKDIR /backend
COPY . /backend
RUN npm install
RUN npx prisma db push
Run npm run build
EXPOSE 8080
CMD ["node","build/index.js"]
~
~
~
~
~
~
~
~
~
~
~
~
~
"Dockerfile" 9L, 164C                                   9,29          All
```

building the image from the docker file

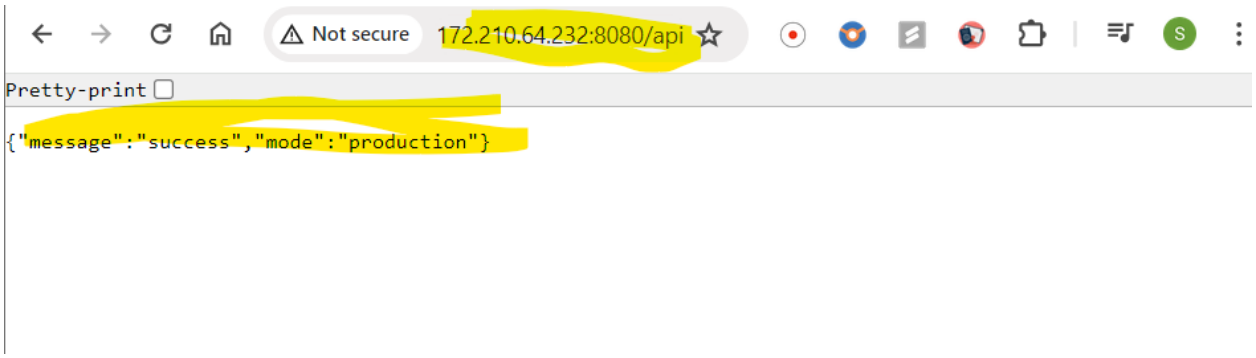
```
azureuser@azure:~/lms/api$ vi Dockerfile
azureuser@azure:~/lms/api$ docker build -t saiteja19799/lms .
[+] Building 39.5s (12/12) FINISHED                          docker:default
 => [internal] load build definition from Dockerfile                    0.0s
 => => transferring dockerfile: 203B                                    0.0s
 => [internal] load metadata for docker.io/library/node:16              0.0s
 => [internal] load .dockerignore                                       0.0s
 => => transferring context: 52B                                        0.0s
 => [1/7] FROM docker.io/library/node:16                                0.0s
 => [internal] load build context                                       0.0s
 => => transferring context: 2.31kB                                     0.0s
 => CACHED [2/7] RUN mkdir /backend                                     0.0s
 => CACHED [3/7] WORKDIR /backend                                       0.0s
 => [4/7] COPY . /backend                                               0.1s
 => [5/7] RUN npm install                                              23.6s
 => [6/7] RUN npx prisma db push                                        4.1s
 => [7/7] RUN npm run build                                             3.7s
 => exporting to image                                                  7.8s
 => => exporting layers                                                 7.8s
 => => writing image sha256:74ae63fb1707ab1e67ab33a113a44d69b2f7e1ee5c158  0.0s
 => => naming to docker.io/saiteja19799/lms                             0.0s
```

Creating a container from the image:

```
azureuser@azure:~/lms/api$ docker container run -dt --name lms-be -p 8080:8080 s
aiteja19799/lms
d8211f52f70387a48a61a41db36ab0f09cfc1afa9b667f7a0d746fc77e607068
```

```
azureuser@azure:~/lms/api$ docker container exec -it lms-be bash
root@d8211f52f703:/backend# pwd
/backend
root@d8211f52f703:/backend# ls
Dockerfile  README.md  build  docker-compose.yml  node_modules  package-lock.json  package.json  prisma  src  tsconfig.json
```

```
                "Aliases": null,
                "MacAddress": "02:42:ac:11:00:04",
                "NetworkID": "8228c582654b059b3b39a8ced060879256dbdc14e8832ead81d5d4554078045b",
                "EndpointID": "930ddc2ab30671dba615a5ab6f8838fdde3c56a388676a291995fa688112b8ba",
                "Gateway": "172.17.0.1",
                "IPAddress": "172.17.0.4",
                "IPPrefixLen": 16,
                "IPv6Gateway": "",
                "GlobalIPv6Address": "",
                "GlobalIPv6PrefixLen": 0,
                "DriverOpts": null,
                "DNSNames": null
            }
        }
    }
}
]
azureuser@azure:~/lms/api$ docker container exec -it lms-be bash
root@d8211f52f703:/backend# curl 172.17.0.4/api
curl: (7) Failed to connect to 172.17.0.4 port 80: Connection refused
root@d8211f52f703:/backend# curl 172.17.0.4:8080/api
{"message":"success","mode":"production"}root@d8211f52f703:/backend#
```



So by this the container is created using image and tested that the backend is building perfectly.

Step 3: Building Docker Image for LMS FRONTEND:

for the frontend the files will be available in webapp folder

First need to connect the front end to back end by giving details of backend into the front end env file

In the webapp folder need to create a docker file

Here we need to give instructions to create folder front end and make it as work directry and then install the node module → then build so that **dist** folder will be created.

Need to transfer the files present in dist folder into nginx folder /usr/share/nginx/html

So that the application would be running on port 80.

Now we can create a container which runs on port 80 then the front end application can be accessed by the user.

azureuser@azure: ~/lms/webapp

```
VITE_API_URL=http://172.210.64.232:8080/api
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
".env" 1L, 44C                                    1,34          All
```

azureuser@azure: ~/lms/webapp

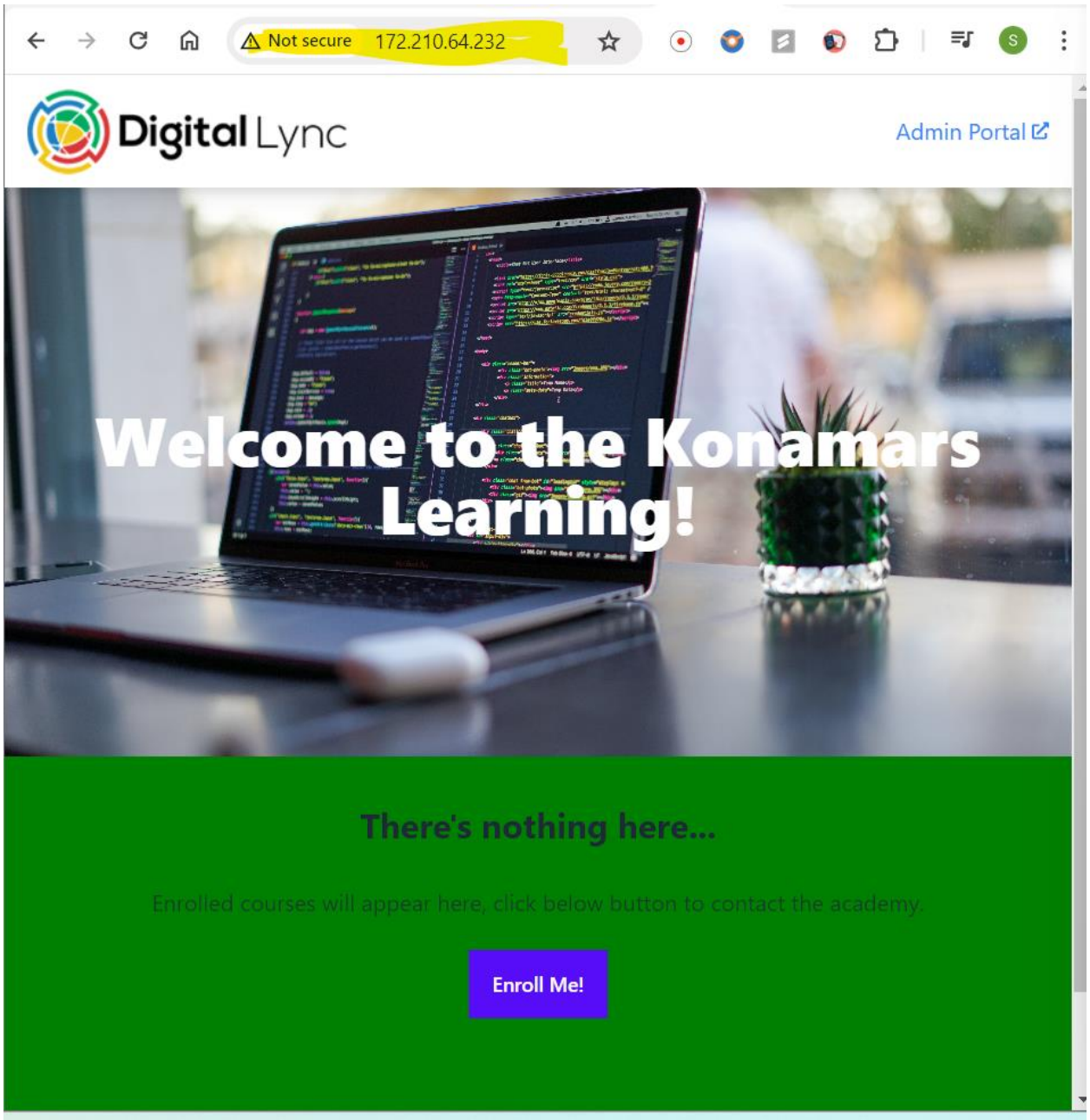
```
FROM node:16 AS fe-build
RUN mkdir /frontend
WORKDIR /frontend
COPY . /frontend
RUN npm install
Run npm run build
FROM nginx
COPY --from=fe-build /frontend/dist /usr/share/nginx/html
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"Dockerfile" 8L, 183C                             8,57          All
```

azureuser@azure: ~/lms/webapp

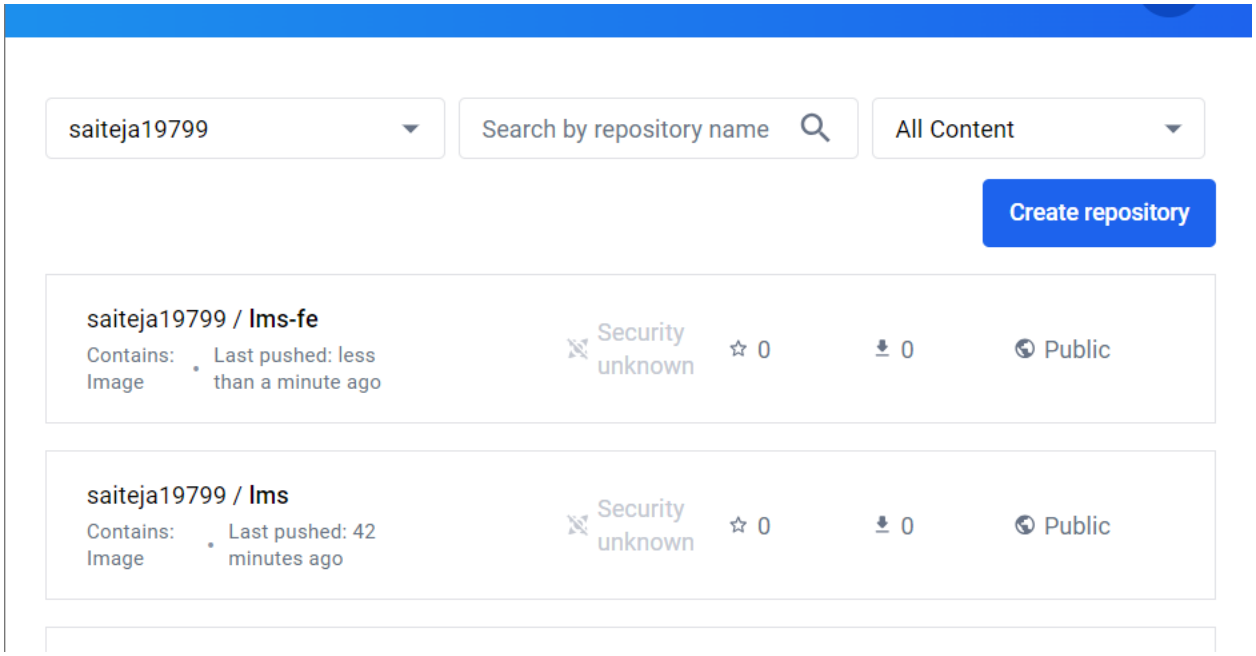
```
Usage:  docker buildx build [OPTIONS] PATH | URL | -

Start a build
azureuser@azure:~/lms/webapp$ docker build -t saiteja19799/lms-fe .
[+] Building 40.2s (15/15) FINISHED                             docker:default
 => [internal] load build definition from Dockerfile                       0.0s
 => => transferring dockerfile: 222B                                       0.0s
 => [internal] load metadata for docker.io/library/nginx:latest            0.2s
 => [internal] load metadata for docker.io/library/node:16                 0.0s
 => [auth] library/nginx:pull token for registry-1.docker.io               0.0s
 => [internal] load .dockerignore                                          0.0s
 => => transferring context: 52B                                           0.0s
 => [fe-build 1/6] FROM docker.io/library/node:16                          0.0s
 => CACHED [stage-1 1/2] FROM docker.io/library/nginx:latest@sha256:a4848  0.0s
 => => resolve docker.io/library/nginx:latest@sha256:a484819eb60211f52990  0.0s
 => [internal] load build context                                          0.0s
 => => transferring context: 4.21kB                                        0.0s
 => CACHED [fe-build 2/6] RUN mkdir /frontend                              0.0s
 => CACHED [fe-build 3/6] WORKDIR /frontend                                0.0s
 => [fe-build 4/6] COPY . /frontend                                        0.1s
 => [fe-build 5/6] RUN npm install                                        22.5s
 => [fe-build 6/6] RUN npm run build                                      16.0s
 => [stage-1 2/2] COPY --from=fe-build /frontend/dist /usr/share/nginx/ht  0.1s
 => exporting to image                                                     0.6s
```

azureuser@azure: ~/lms/webapp

```
 => => naming to docker.io/saiteja19799/lms-fe                             0.0s
azureuser@azure:~/lms/webapp$ docker container run -dt --name f2 saiteja19799/lm
s-fe
33d32ad8280b7b6683ae6354351f0a46f6efe8c641e876ca585d3a0d0bf605f0
azureuser@azure:~/lms/webapp$ docker container exec -it f2 bash
root@33d32ad8280b:/# ls /usr/share/nginx/html
50x.html  assets  dl-logo.png  index.html  mockServiceWorker.js  vite.svg
root@33d32ad8280b:/# exit
exit
azureuser@azure:~/lms/webapp$ docker container inspect f2
[
    {
        "Id": "33d32ad8280b7b6683ae6354351f0a46f6efe8c641e876ca585d3a0d0bf605f0"
,
        "Created": "2024-05-23T04:25:09.383415071Z",
        "Path": "/docker-entrypoint.sh",
        "Args": [
            "nginx",
            "-g",
            "daemon off;"
        ],
        "State": {
            "Status": "running",
            "Running": true,
```



So we have created images for the frontend end and backend so we can use these images to build the containers and build the application within a minute.



The images were pushed into docker hub so now we can run the application in any machine by creating the containers using the images in docker hub.