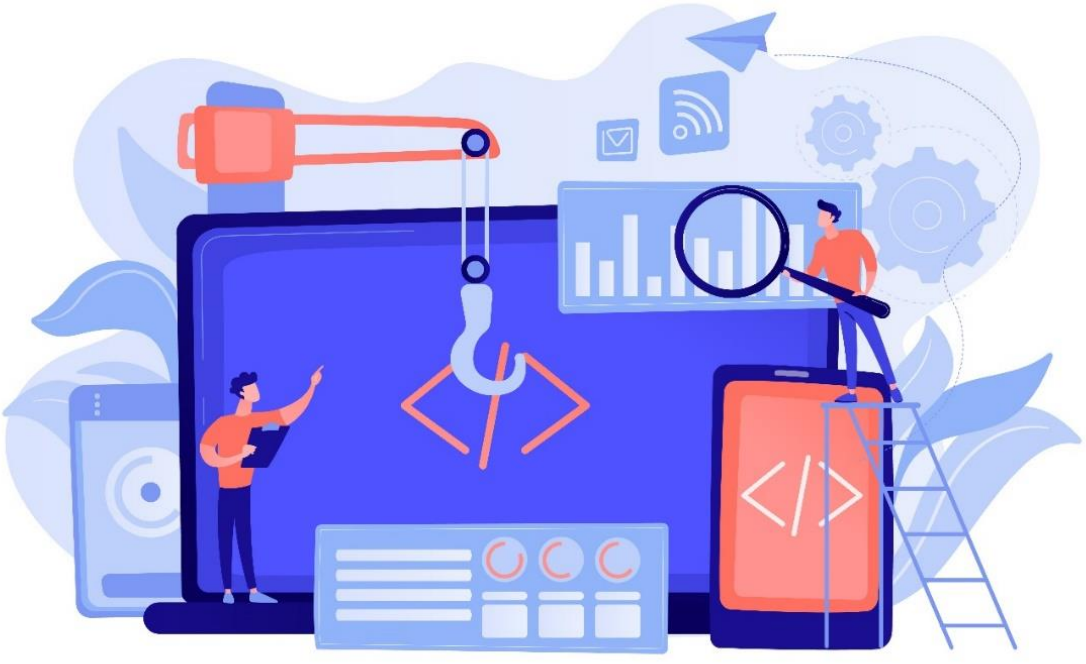
المعهد الوطني للبريد والمواصلات
Institut National des Postes et Télécommunications

الوكالة الوطنية لتقنين المواصلات
Agence Nationale de Réglementation des Télécommunications

Rapport : Exam (Mini-Projet) | **Encadré par :** Prof. Driss ALLAKI

Conteneurisation des Applications



Youssef Zahi - ASEDS

Année universitaire : 2022/2023

Part One

1)

- In the process of creating the mongodb-service container, I persist the container's data using volumes, I choose the Docker-managed volumes, because they are easy to remove and they bind only to a single container, also the docker managed volumes are independent from the host.

- To create this container, I used the command below:

docker run -d -name mongodb-service -v /data/mongodb -p 27017:27017 mongo

```
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project> docker run -d --name mongodb-service -v /data/mongodb -p 27017:27017 mongo
Unable to find image 'mongo:latest' locally
latest: Pulling from library/mongo
eaead16dc43b: Pull complete
8a00eb9f68a0: Pull complete
f683956749c5: Pull complete
b33b2f05ea20: Pull complete
3a342bea915a: Pull complete
fa956ab1c2f0: Pull complete
138a8542a624: Pull complete
acab179a7f07: Pull complete
f88335710e84: Pull complete
Digest: sha256:71a63fc2438e45714f6c8a2505968ee0beeb94ec77a88ef12190f7cee9b95f32
Status: Downloaded newer image for mongo:latest
7fd2cbb9b2ca293cfdeb2a37d861c26bee70ed5d662e223a42757b0655e87e1b
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project>
```

2)

- Creating docker network:

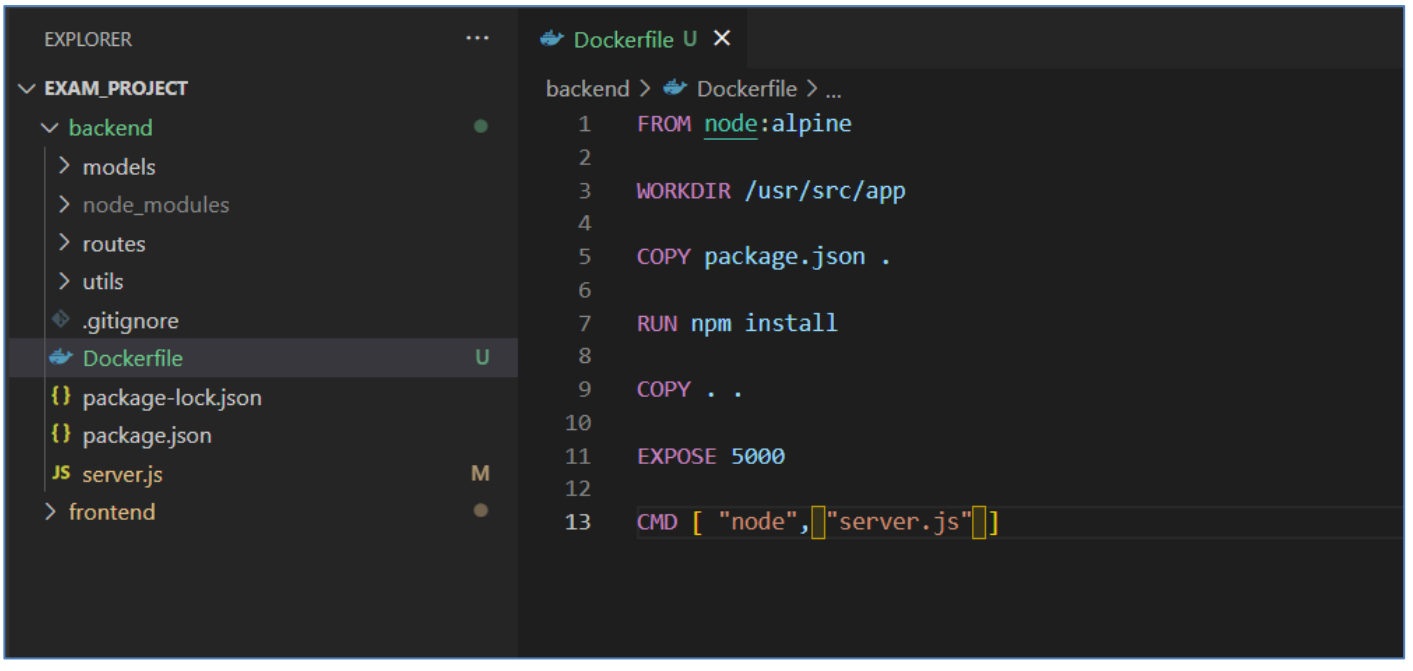
```
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project> docker network create --driver bridge examNetwork
a80fd8475d8d72462506f99169bfabb126c0b8e71c5453d5dcbb85d6c3853581
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project> docker network ls
NETWORK ID     NAME          DRIVER    SCOPE
91b9f8aaffee   bridge        bridge    local
a80fd8475d8d   examNetwork   bridge    local
9d9ec2b0616e   host          host      local
d5e0ff1a675e   none          null      local
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project>
```

- For connecting the mongodb-service container to this network we used the command below:

```
eaead16dc43b: Pull complete
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project> docker network connect examNetwork mongodb-service
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project> docker network inspect examNetwork
[
    {
        "Name": "examNetwork",
        "Id": "a80fd8475d8d72462506f99169bfabb126c0b8e71c5453d5dcbb85d6c3853581",
        "Created": "2022-11-13T21:52:05.878523Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "7fd2cbb9b2ca293cfdeb2a37d861c26bee70ed5d662e223a42757b0655e87e1b": {
                "Name": "mongodb-service",
                "EndpointID": "a0a5ba1571390181f6cd8bc5fd5620ed7b53a07bb6c845ea982b35d7539464fc",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project>
```

3)

- Creating Docker file for the backend project:



4)

- The best practices used for writing this Docker files are:
  - WORKDIR
    - For clarity and reliability, we should always use absolute paths for our WORKDIR

- Use Specific Images Tags
  - When choosing the base image, it's recommended to use a specific tag. We should avoid using the latest tag for the image, because the latest tag can have breaking changes over time. Because those changes may create conflicts between co-workers.
- Leverage build cache
  - When building an image, Docker steps through the instructions in your Dockerfile, executing each in the order specified. As each instruction is examined, Docker looks for an existing image in its cache that it can reuse, rather than creating a new (duplicate) image.
- Remove Unneeded Dependencies
  - Docker images should be as minimal as possible.
- Use alpine version
  - Images alpine version provides a lightweight and popular Docker image that can improve our image build and deployment.

5)

a)

- Creating local docker image using the backend Dockerfile

```
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\backend> docker build -t backend-image .
[+] Building 23.4s (5/10)
 => [internal] load build definition from Dockerfile                                                                    0.1s
 => => transferring dockerfile: 182B                                                                                    0.0s
 => [internal] load .dockerignore                                                                                       0.0s
 => => transferring context: 2B                                                                                         0.0s
 => [internal] load metadata for docker.io/library/node:18.12.1                                                         4.2s
 => [auth] library/node:pull token for registry-1.docker.io                                                             0.0s
 => [1/5] FROM docker.io/library/node:18.12.1@sha256:c47a2c61e635eb4938fcd56a1139b552300624b53e3eca06b5554a577f1842cf   18.9s
 => => resolve docker.io/library/node:18.12.1@sha256:c47a2c61e635eb4938fcd56a1139b552300624b53e3eca06b5554a577f1842cf    0.0s
 => => sha256:c47a2c61e635eb4938fcd56a1139b552300624b53e3eca06b5554a577f1842cf 1.21kB / 1.21kB                          0.0s
 => => sha256:2771803756cf54d0b8031fa5239420386608bcff9f69f9e8a7afda0671982537 2.21kB / 2.21kB                          0.0s
 => => sha256:a8ca11554fce00d9177da2d76307bdc06df7faeb84529755c648ac4886192ed1 51.38MB / 55.04MB                       18.9s
 => => sha256:c85a0be79bfba309d1f05dc40b447aa82b604593531fed1e7e12e4bef63483a5 10.88MB / 10.88MB                        2.3s
 => => sha256:e390ceb997819ddf74b976b6a744ad56afbc5f680ab9bcec68036815f504b010 7.51kB / 7.51kB                          0.0s
 => => sha256:e4e46864aba2e62ba7c75965e4aa33ec856ee1b1074dda6b478101c577b63abd 5.16MB / 5.16MB                          9.8s
 => => sha256:195ea6a58ca87a18477965a6e6a8623112bde82c5b568a29c56ce4581b6e6695 35.65MB / 54.59MB                       18.9s
 => => sha256:157f16ed0a0c119e5015d22d95fd158bf9e85654611b870b79cca3987442948b 14.68MB / 196.87MB                      18.9s
 => [internal] load build context                                                                                       3.1s
 => => transferring context: 187.77kB                                                                                   2.9s
```

```
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\backend> docker images
REPOSITORY                                        TAG                                                                IMAGE ID       CREATED        SIZE
backend-image                                     latest                                                             5f4faa47a16b   2 minutes ago  1.04GB
mongo                                             latest                                                             b70536aeb250   3 weeks ago    695MB
quay.io/prometheus-operator/prometheus-config-reloader   v0.60.1                                                    6dd13479f484   5 weeks ago    12.2MB
quay.io/prometheus/prometheus                     v2.39.1                                                            6b9895947e9e   5 weeks ago    220MB
hubproxy.docker.internal:5000/docker/desktop-kubernetes   kubernetes-v1.25.0-cni-v1.1.1-critools-v1.24.2-cri-dockerd-v0.2.5-1-debian   2042e761d17a   2 months ago   363MB
k8s.gcr.io/kube-apiserver                         v1.25.0                                                            4d2edfd10d3e   2 months ago   128MB
k8s.gcr.io/kube-controller-manager                v1.25.0                                                            1a54c86c03a6   2 months ago   117MB
k8s.gcr.io/kube-scheduler                         v1.25.0                                                            bef2cf311509   2 months ago   50.6MB
k8s.gcr.io/kube-proxy                             v1.25.0                                                            58a9a0c6d96f   2 months ago   61.7MB
k8s.gcr.io/pause                                  3.8                                                                4873874c08ef   5 months ago   711kB
k8s.gcr.io/etcd                                   3.5.4-0                                                            a8a176a5d5d6   5 months ago   300MB
k8s.gcr.io/ingress-nginx/controller               <none>                                                             75bdf78d9d67   5 months ago   289MB
k8s.gcr.io/coredns                                v1.9.3                                                             5185b96f0bec   5 months ago   48.8MB
quay.io/prometheus/alertmanager                   v0.24.0                                                            e556bd45e16c   7 months ago   59.7MB
quay.io/prometheus/node-exporter                  v1.3.1                                                             1dbe0e931976   11 months ago  20.9MB
docker/desktop-vpnkit-controller                  v2.0                                                               8c2c38aa676e   18 months ago  21MB
docker/desktop-storage-provisioner                v2.0                                                               99f89471f470   18 months ago  41.9MB
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\backend>
```

"-t (tag)": Identifying the image with its tag.
".": For specifying the relative path of Dockerfile.

b)

- Scanning the image of vulnerabilities it may contains using the command: " docker scan imageName "

```
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\backend> docker scan backend-image

Testing backend-image...

Package manager:   apk
Project name:      docker-image|backend-image
Docker image:      backend-image
Platform:          linux/amd64
Base image:        node:18.12.1-alpine3.16

✓ Tested 16 dependencies for known vulnerabilities, no vulnerable paths found.

According to our scan, you are currently using the most secure version of the selected base image

For more free scans that keep your images secure, sign up to Snyk at https://dockr.ly/3ePqVcp

-------------------------------------------------------

Testing backend-image...

Tested 143 dependencies for known vulnerabilities, found 24 vulnerabilities.
```

c) Publish the image in Docker Hub:

- First, we should create a repository in Docker Hub, then connect to our account via CLI and finally push it to the docker hub
- We use these two commands:

- docker tag imageName: tagName userName/repoName
- docker push username/repoName

```
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\backend> docker tag backend-image youssefzahi/exam_project
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\backend> docker push youssefzahi/exam_project
Using default tag: latest
The push refers to repository [docker.io/youssefzahi/exam_project]
3fce86b17057: Pushed
9d854413a774: Pushed
8bfc581860c8: Layer already exists
a48011d48973: Layer already exists
b2d2930f5207: Layer already exists
3d3b9564a8d2: Layer already exists
0d519f3bcae3: Pushed
e5e13b0c77cb: Layer already exists
latest: digest: sha256:f32530467c06da01535c513453bb5ebe4b4e606c3ea74f619a728b89ac2a43da size: 1993
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\backend>
```

d) Running the backend container

```
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\backend> docker run --name backend -p 5000:5000 --network examNetwork -d backend-image
f4ed1af33b06b94a2253eb1b42fc17ed505a5b004766d6807e6097276b6f954d
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\backend>
```

--name: to name the container
--network: specifying the network name
-d: to run the container in detached mode
-p: for port mapping

e)

- Inspect the backend container using the command :

" docker inspect containerName "

```
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\backend> docker inspect backend
[
    {
        "Id": "f4ed1af33b06b94a2253eb1b42fc17ed505a5b004766d6807e6097276b6f954d",
        "Created": "2022-11-16T21:14:42.4989707Z",
        "Path": "docker-entrypoint.sh",
        "Args": [
            "/bin/sh",
            "-c",
            "[ \"node\", \"server.js\" ]"
        ],
        "State": {
            "Status": "exited",
            "Running": false,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 0,
            "ExitCode": 2,
            "Error": "",
            "StartedAt": "2022-11-16T21:14:44.5637557Z",
            "FinishedAt": "2022-11-16T21:14:44.6074235Z"
```

f)

- List the backend container logs using the following command: "docker logs containerName"

```
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\backend> docker logs backend
Server v8 up and running on port 5000 !
MongoDB successfully connected
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\backend>
```

6)

- To redo the work requested in questions 3), 4) and 5) for the frontend project:

1. Firstly, we create frontend Dockerfile:
    - We will use docker multi-stage builds. With that, we can have a Node.js base image that installs, builds and compiles everything, and then, “discard” all those Node.js specific Docker image layers, and we will end up with a Nginx image with just the compiled code.

Dockerfile U

frontend > Dockerfile > ...

```dockerfile
# Stage 1
FROM node:16-alpine3.12 AS builder

ENV CLOUDL_SERVER="localhost"

WORKDIR /src

COPY package.json .

RUN npm install

COPY . .

RUN npm run build

# Stage 2
FROM nginx:1.23.2-alpine

COPY --from=builder /src/build   /usr/share/nginx/html

COPY  ./nginx/nginx.conf   /etc/nginx/conf.d/default.conf

EXPOSE 80

CMD ["nginx", "-g","daemon off;"]
```

2. Secondly, we will build the image:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER COMMENTS

```
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\frontend> docker build -t frontend-image .
[+] Building 1.5s (15/15) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 421B
 => [internal] load .dockerignore
 => => transferring context: 34B
 => [internal] load metadata for docker.io/library/nginx:1.23.2-alpine
 => [internal] load metadata for docker.io/library/node:16-alpine3.12
 => [internal] load build context
 => => transferring context: 1.80kB
 => [builder 1/6] FROM docker.io/library/node:16-alpine3.12@sha256:c2ed3b2b36b726980474f8bf80025ca3a1aeb90c76286953f9f4b9b1dc3001b0
 => [stage-1 1/3] FROM docker.io/library/nginx:1.23.2-alpine@sha256:455c39afebd4d98ef26dd70284aa86e6810b0485af5f4f222b19b89758cabf1e
 => CACHED [builder 2/6] WORKDIR /src
 => CACHED [builder 3/6] COPY package.json .
 => CACHED [builder 4/6] RUN npm install
 => CACHED [builder 5/6] COPY . .
 => CACHED [builder 6/6] RUN npm run build
 => CACHED [stage-1 2/3] COPY --from=builder /src/build   /usr/share/nginx/html
 => [stage-1 3/3] COPY  ./nginx/nginx.conf   /etc/nginx/conf.d/default.conf
 => exporting to image
 => => exporting layers
 => => writing image sha256:30c82a94463d1c04c6a8293901da9dcc15f550f3a96bdcc9d7f077adde64a67b
 => => naming to docker.io/library/frontend-image
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\frontend>
```

3. Thirdly, we will scan the vulnerabilities

```
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\frontend> docker scan frontend-image

Testing frontend-image...

Package manager:   apk
Project name:      docker-image|frontend-image
Docker image:      frontend-image
Platform:          linux/amd64
Base image:        nginx:1.23.2-alpine

✓ Tested 43 dependencies for known vulnerabilities, no vulnerable paths found.

According to our scan, you are currently using the most secure version of the selected base image

For more free scans that keep your images secure, sign up to Snyk at https://dockr.ly/3ePqVcp


PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\frontend>
```

4. Fourthly, we will push our image to docker hub:

```
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\frontend> docker tag frontend-image youssefzahi/frontend-image
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\frontend> docker push youssefzahi/frontend-image
Using default tag: latest
The push refers to repository [docker.io/youssefzahi/frontend-image]
97fc01a13727: Pushed
d5f3bdae4bfd: Pushed
bd502c2dee4c: Mounted from library/nginx
9365b1fffb04: Mounted from library/nginx
6636f46e559d: Mounted from library/nginx
fcf860bf48b4: Mounted from library/nginx
07099189e7ec: Mounted from library/nginx
e5e13b0c77cb: Mounted from youssefzahi/exam_project
latest: digest: sha256:7a6f6c2e3dcc73920e3f467f5e9fc3cd27d275b66e6dd4da3578f42cd91aa25f size: 1986
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\frontend>
```

5. Fifthly, we will create the frontend project container

```
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\frontend> docker run -d --name frontend -p 80:80 --network examNetwork frontend-image
5bf1d9ed28ac1814a647bd5dceaccd74945e3eabd85f459b204f6806909a89dd
```

6. Sixthly, we will inspect the frontend container:

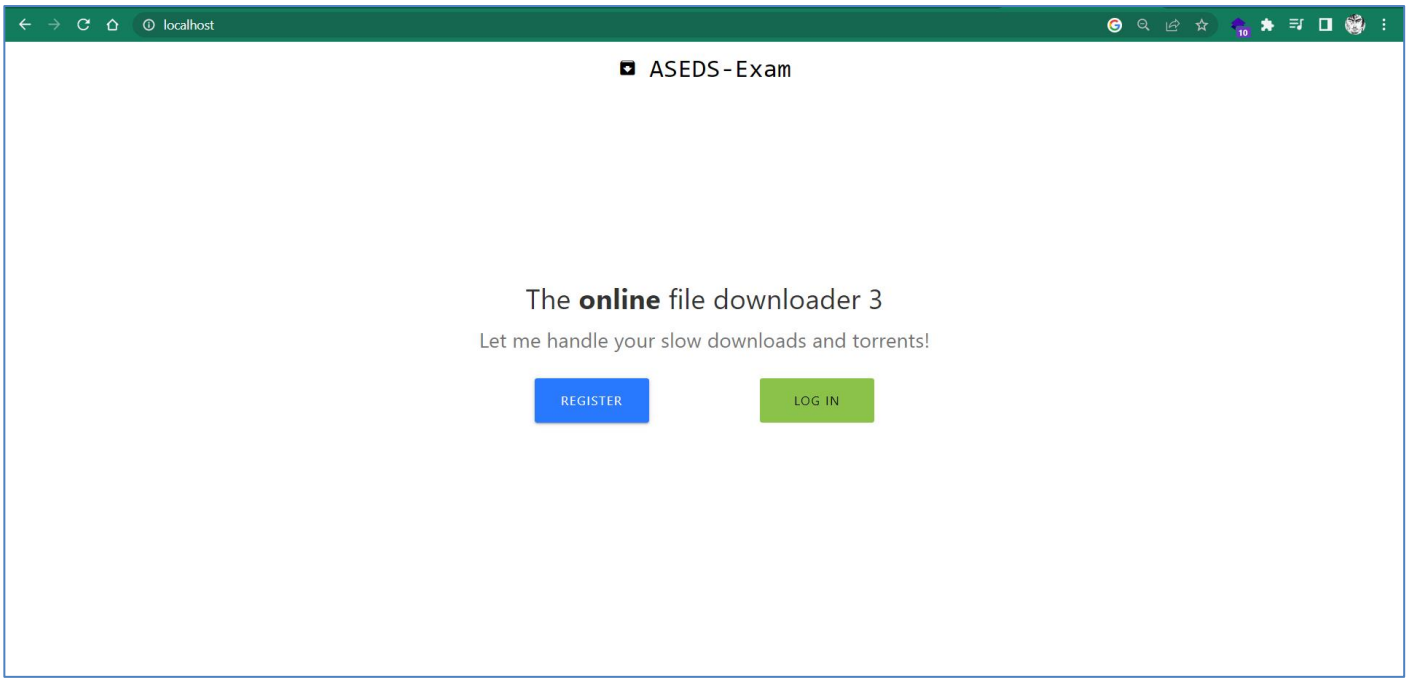
```
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\frontend> docker inspect frontend
[
    {
        "Id": "5bf1d9ed28ac1814a647bd5dceaccd74945e3eabd85f459b204f6806909a89dd",
        "Created": "2022-11-20T09:44:00.1953913Z",
        "Path": "/docker-entrypoint.sh",
        "Args": [
            "nginx",
            "-g",
            "daemon off;"
        ],
        "State": {
            "Status": "exited",
            "Running": false,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 0,
            "ExitCode": 1,
            "Error": "",
            "StartedAt": "2022-11-20T09:44:02.3737124Z",
            "FinishedAt": "2022-11-20T09:44:06.1801092Z"
        },
        "Image": "sha256:30c82a94463d1c04c6a8293901da9dcc15f550f3a96bdcc9d7f077adde64a67b",
        "ResolvConfPath": "/var/lib/docker/containers/5bf1d9ed28ac1814a647bd5dceaccd74945e3eabd85f459b204f6806909a89dd/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/5bf1d9ed28ac1814a647bd5dceaccd74945e3eabd85f459b204f6806909a89dd/hostname",
        "HostsPath": "/var/lib/docker/containers/5bf1d9ed28ac1814a647bd5dceaccd74945e3eabd85f459b204f6806909a89dd/hosts",
```

7. Finally, we will list the backend container logs

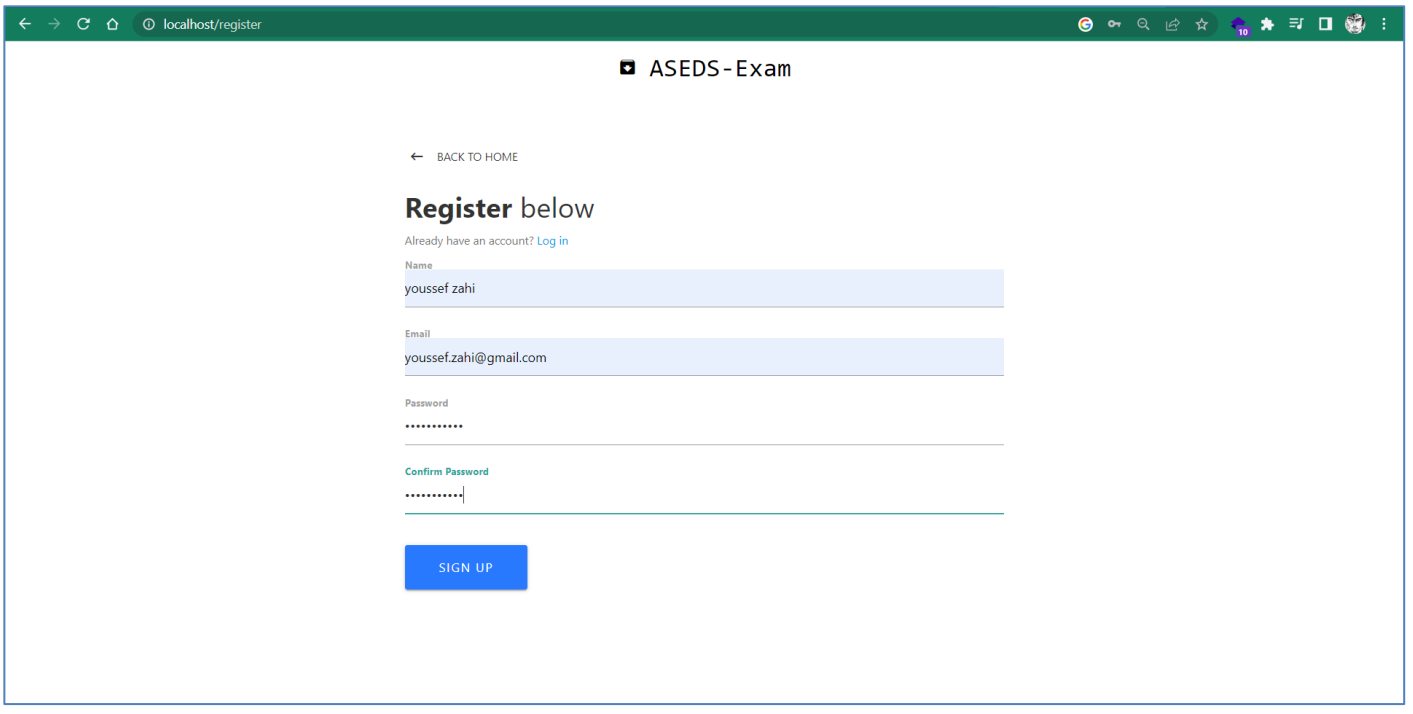
```
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\frontend> docker logs frontend
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf differs from the packaged version
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2022/11/20 09:56:35 [notice] 1#1: using the "epoll" event method
2022/11/20 09:56:35 [notice] 1#1: nginx/1.23.2
2022/11/20 09:56:35 [notice] 1#1: built by gcc 11.2.1 20220219 (Alpine 11.2.1_git20220219)
2022/11/20 09:56:35 [notice] 1#1: OS: Linux 5.10.16.3-microsoft-standard-WSL2
2022/11/20 09:56:35 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2022/11/20 09:56:35 [notice] 1#1: start worker processes
2022/11/20 09:56:35 [notice] 1#1: start worker process 29
2022/11/20 09:56:35 [notice] 1#1: start worker process 30
2022/11/20 09:56:35 [notice] 1#1: start worker process 31
2022/11/20 09:56:35 [notice] 1#1: start worker process 32
2022/11/20 09:56:35 [notice] 1#1: start worker process 33
2022/11/20 09:56:35 [notice] 1#1: start worker process 34
2022/11/20 09:56:35 [notice] 1#1: start worker process 35
2022/11/20 09:56:35 [notice] 1#1: start worker process 36
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\frontend>
```

7)

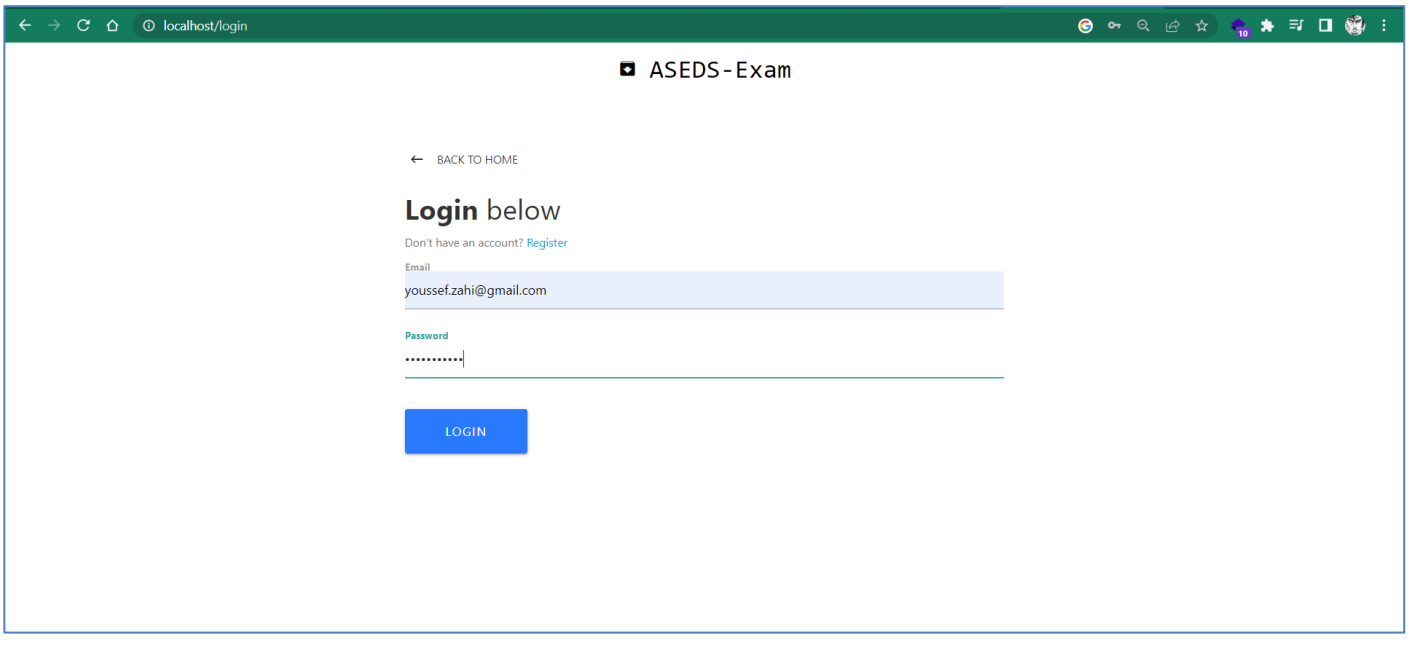
- Testing the application



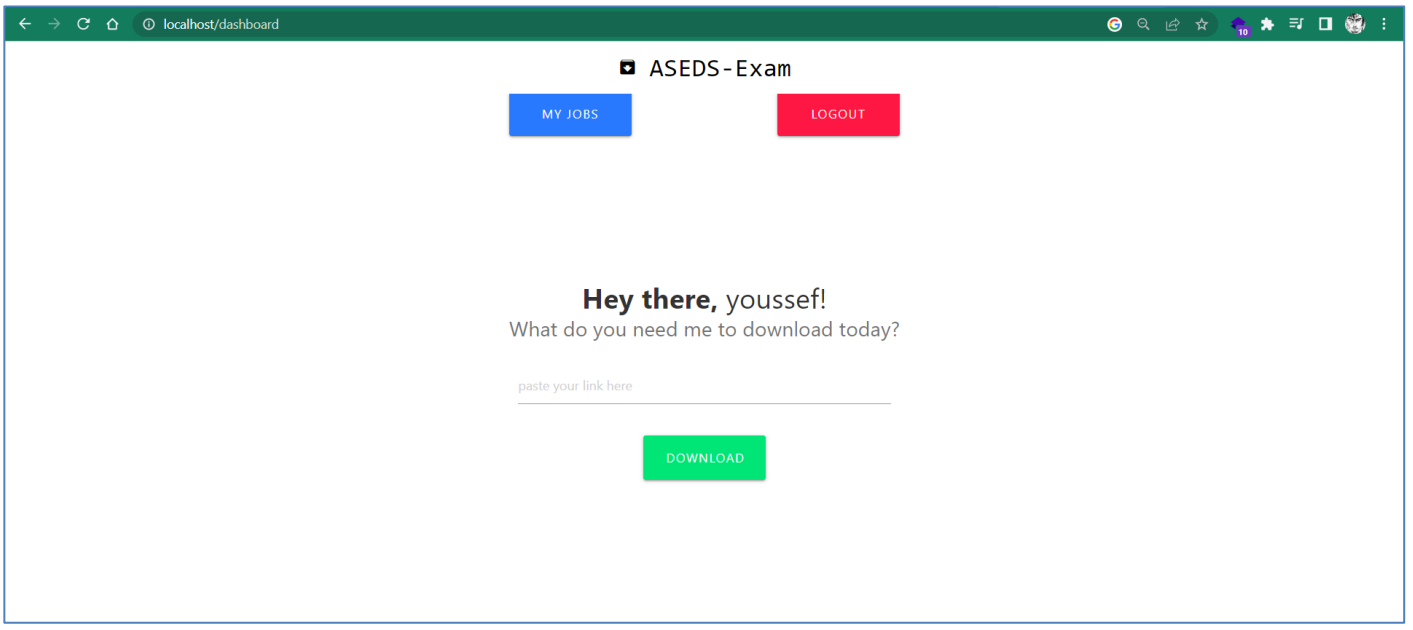
- Register screen



- Sign in screen



- User home screen



8)

- Removing the containers :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    JUPYTER    COMMENTS

PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project> docker stop frontend backend mongodb-service
frontend
backend
mongodb-service
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project> docker rm  frontend backend mongodb-service
frontend
backend
mongodb-service
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project>
```

9)

- Redeploy the application using docker-compose
  - a)
    - Create the docker-compose file

docker-compose.yaml

```yaml
version: '2.10.2'
services:
  mongodb-service:
    image: mongo
    volumes:
      - mongodbdata:/data/mongodb
    container_name: mongodb-service
    ports:
      - 27017:27017
    networks:
      - examNetwork

  backend:
    image: youssefzahi/backend-img
    container_name: backend
    ports:
      - 5000:5000
    networks:
      - examNetwork
    depends_on:
      - mongodb-service
```

docker-compose.yaml

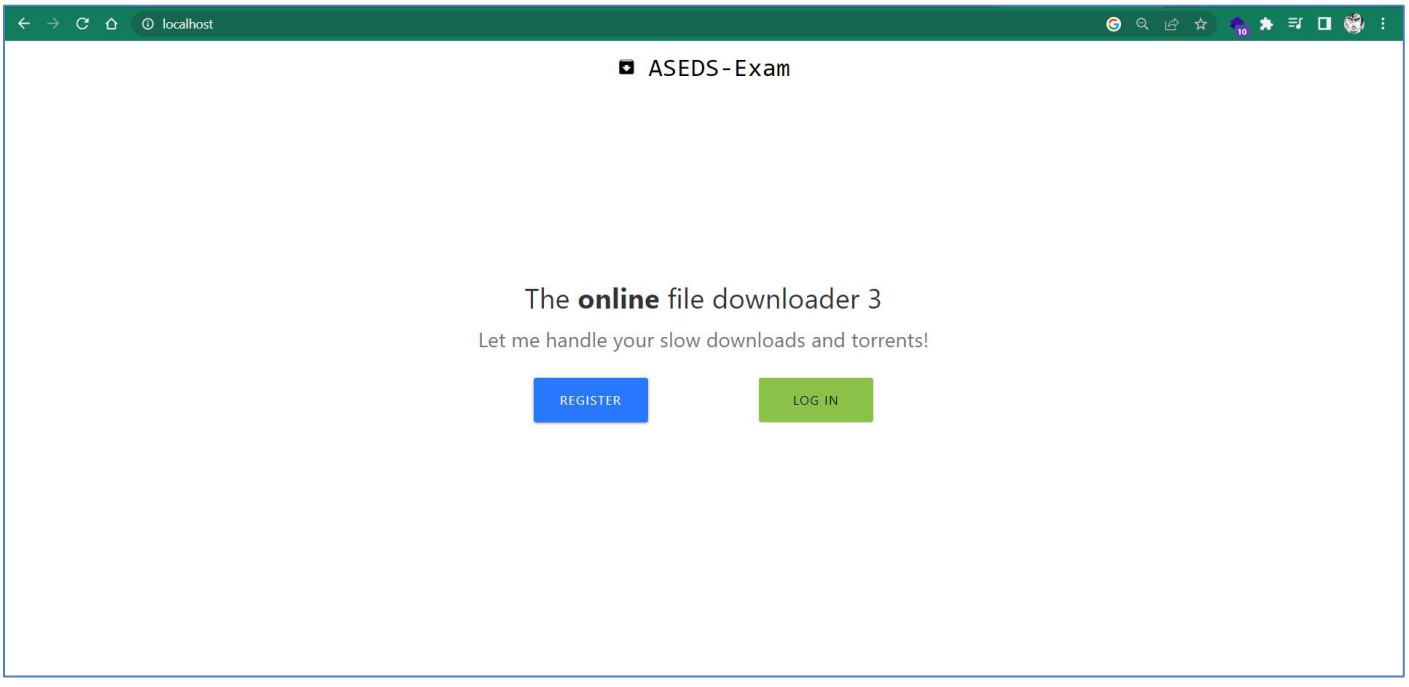
```yaml
  frontend:
    image: youssefzahi/frontend-img
    container_name: frontend
    ports:
      - 80:80
    networks:
      - examNetwork
    depends_on:
      - backend
    # allows to run the application with
    # it flag which is required with react
    stdin_open: true
    tty: true

volumes:
  mongodbdata:

networks:
  examNetwork:
    external: true
```

b)

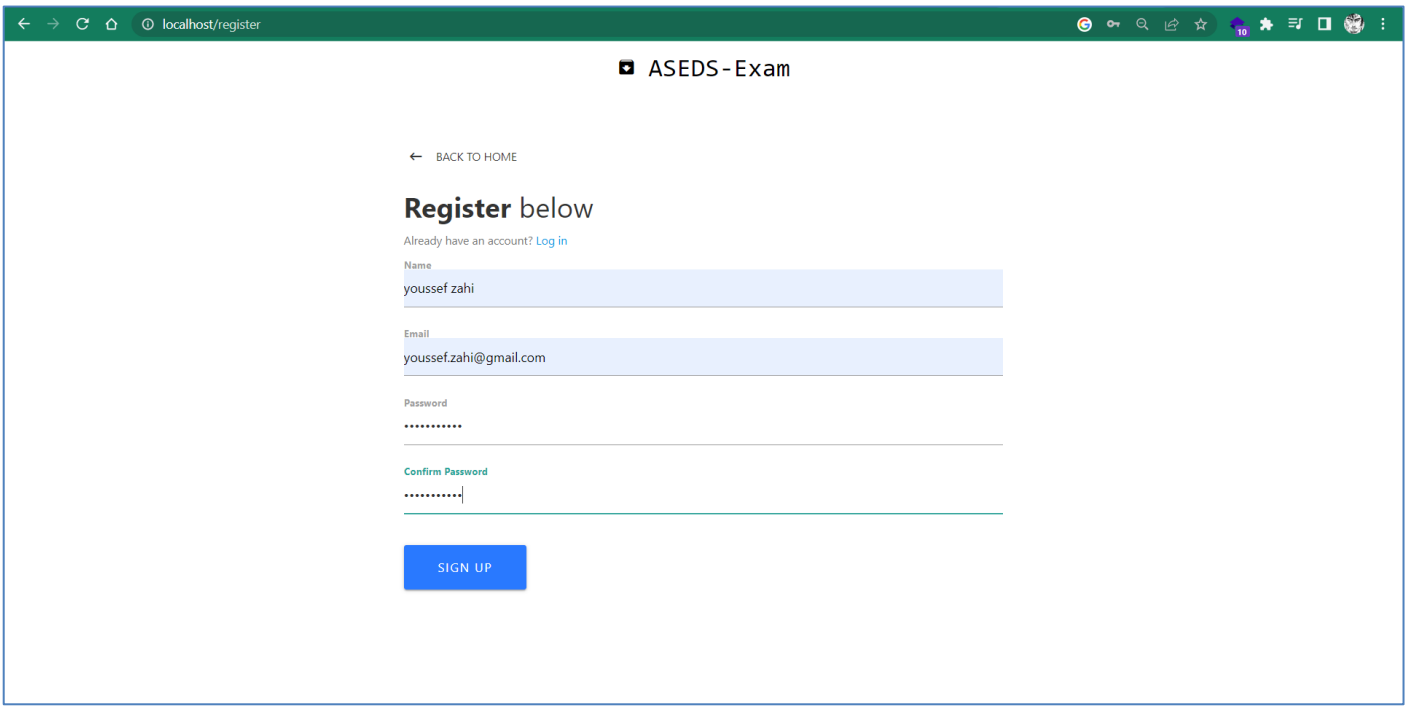
- After create the docker-compose file, we will run it using the command: "**docker-compose up**"

c)

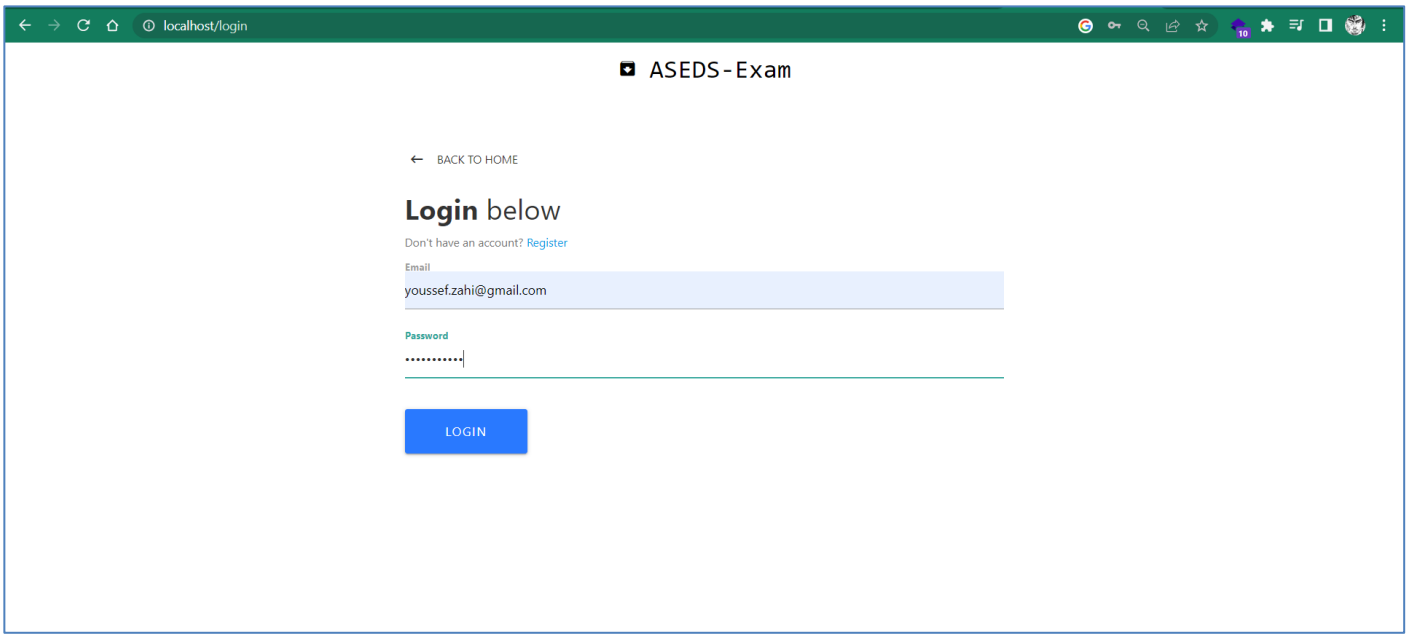
- Testing the application



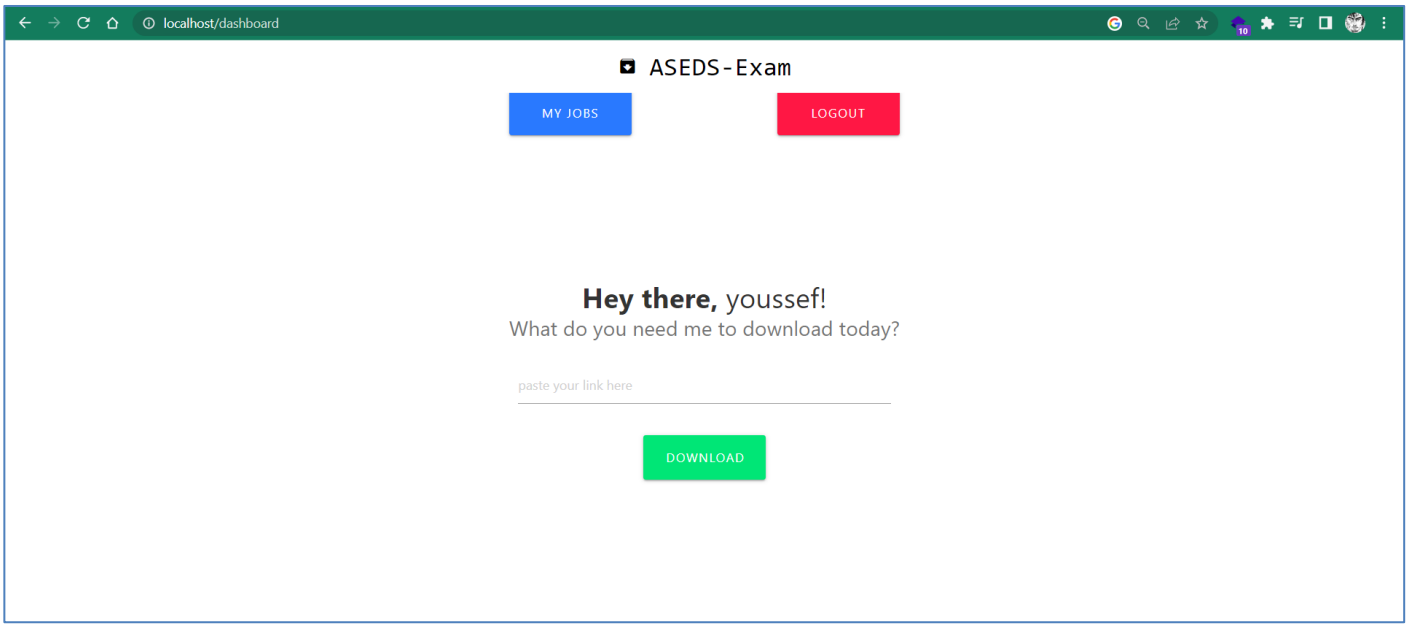
- Register Screen



- Sign in screen



- User home screen



10)

- Stopping the docker containers using: "docker-compose down"

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    JUPYTER    COMMENTS

PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project\frontend> docker-compose down
[+] Running 3/3
 - Container frontend         Removed
 - Container backend          Removed
 - Container mongodb-service  Removed
```

11)

- Creating a private registry:
  - Firstly, we create a docker compose registry file:

docker-compose-registry.yaml U

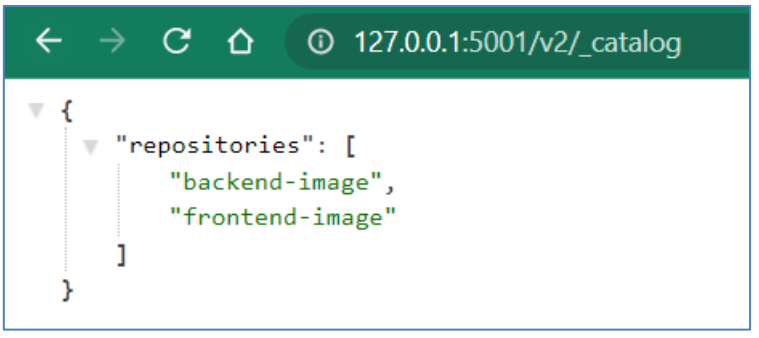
docker-compose-registry.yaml

```yaml
version: '3'
services:
  docker-registry:
    image: registry:2
    container_name: docker-registry
    restart: always
    ports:
    - "5001:5000"
    volumes:
      - ./volume:/var/lib/registry

  docker-registry-ui:
    image: konradkleine/docker-registry-frontend:v2
    container_name: docker-registry-ui
    restart: always
    ports:
    - "8080:80"
    environment:
        ENV_DOCKER_REGISTRY_HOST: docker-registry
        ENV_DOCKER_REGISTRY_PORT: 5000
```

a)

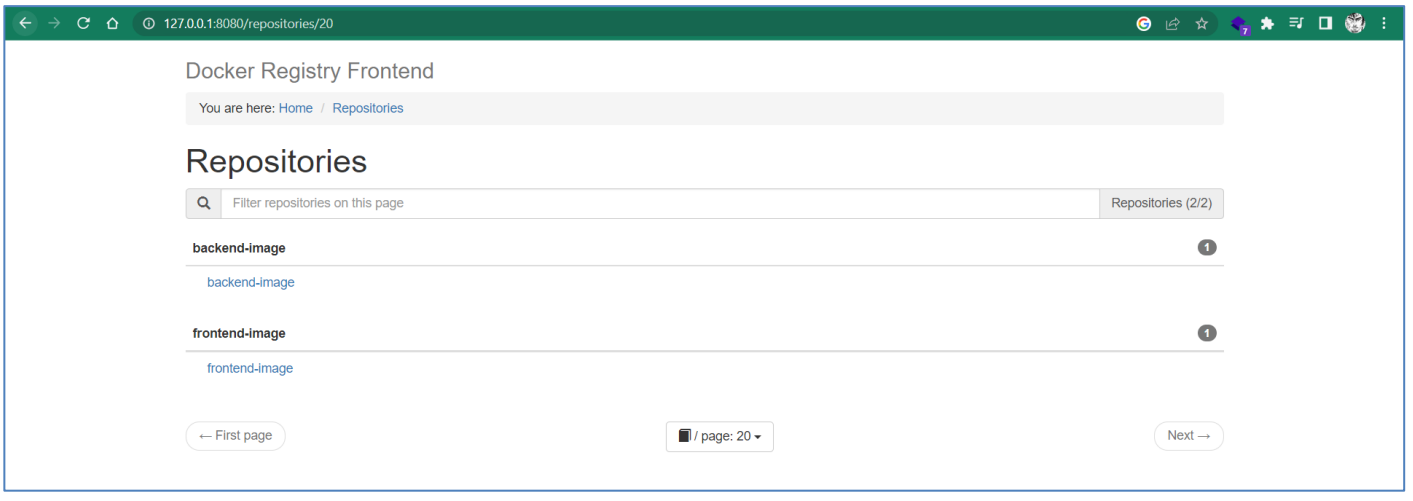
- After creating the registry, now we will push the existing images that we built to it.

```
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project> docker tag backend-image 127.0.0.1:5001/backend-image
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project> docker push 127.0.0.1:5001/backend-image
Using default tag: latest
The push refers to repository [127.0.0.1:5001/backend-image]
e927817f0030: Pushed
c0052cc73372: Pushed
c4e64f125cc2: Pushed
a48011d48973: Pushed
b2d2930f5207: Pushed
3d3b9564a8d2: Pushed
0d519f3bcae3: Pushed
e5e13b0c77cb: Pushed
latest: digest: sha256:abfcb74d7c4698bd19ccbc96beca550424a9f157520bd89ab47c07460a0f15e8 size: 1993
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project> docker tag frontend-image 127.0.0.1:5001/frontend-image
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project> docker push 127.0.0.1:5001/frontend-image
Using default tag: latest
The push refers to repository [127.0.0.1:5001/frontend-image]
a0cd3dfa9797: Pushed
d5f3bdae4bfd: Pushed
bd502c2dee4c: Pushed
9365b1fffb04: Pushed
6636f46e559d: Pushed
fcf860bf48b4: Pushed
07099189e7ec: Pushed
e5e13b0c77cb: Mounted from backend-image
latest: digest: sha256:b8f6ec5b6f5c67848373d6bb4eb36ba49168852eb9baa8dc1933195a0e36c280 size: 1986
PS C:\Users\youss\Documents\ASEDS\S3\P1\Conterization\exam_project>
```



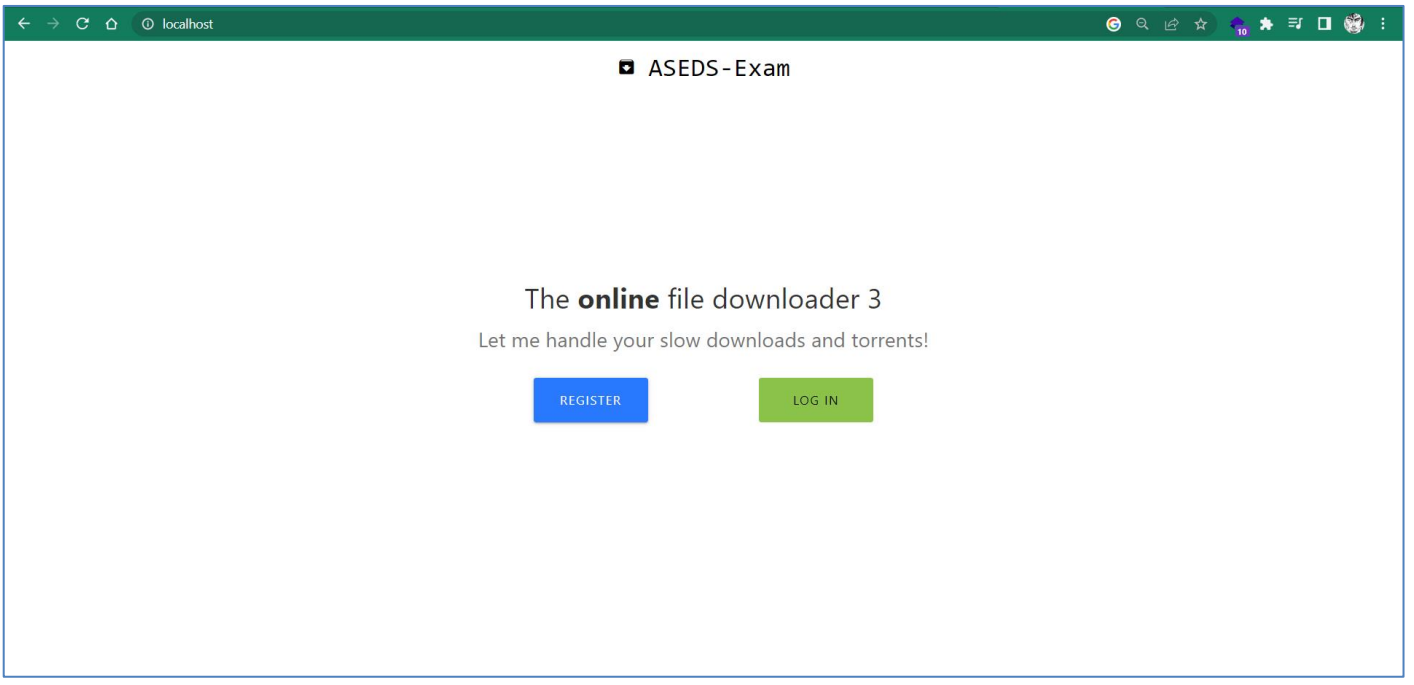
b)

- To visualize the images in the registry, we will visit the localhost:8080/home

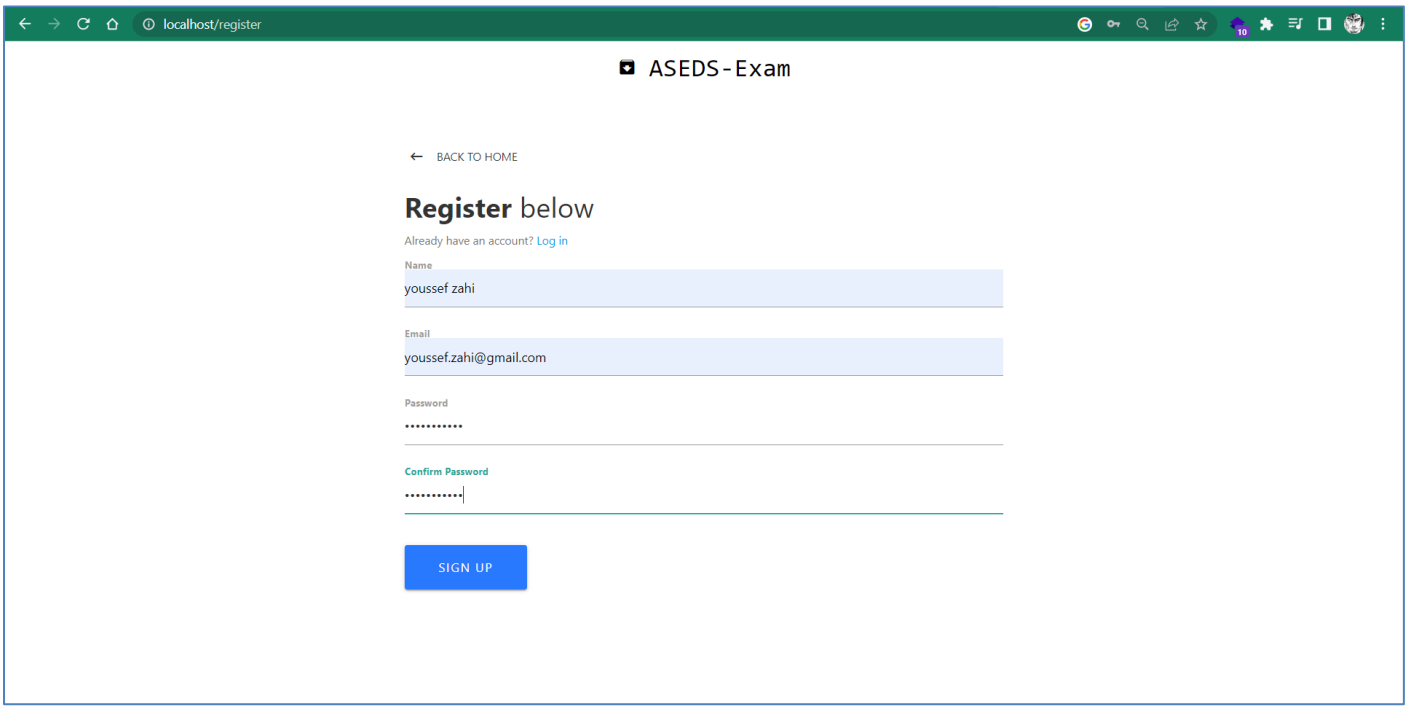


11)

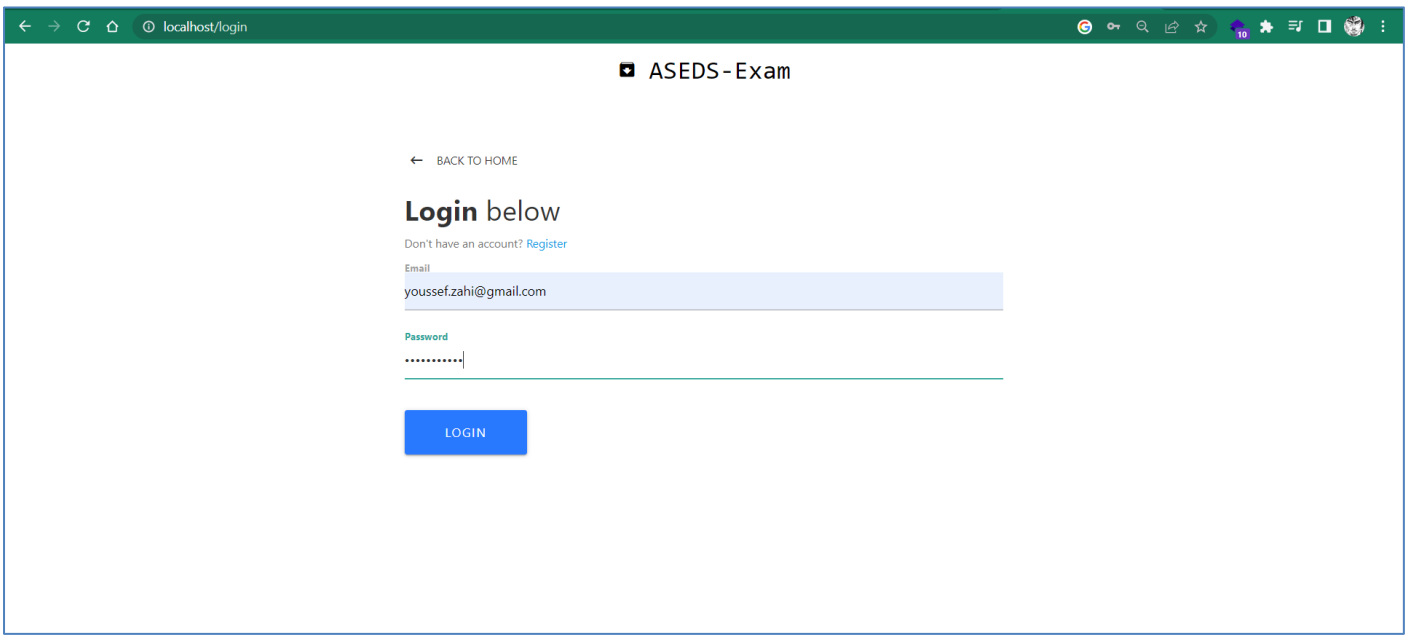
- The application test passed successfully



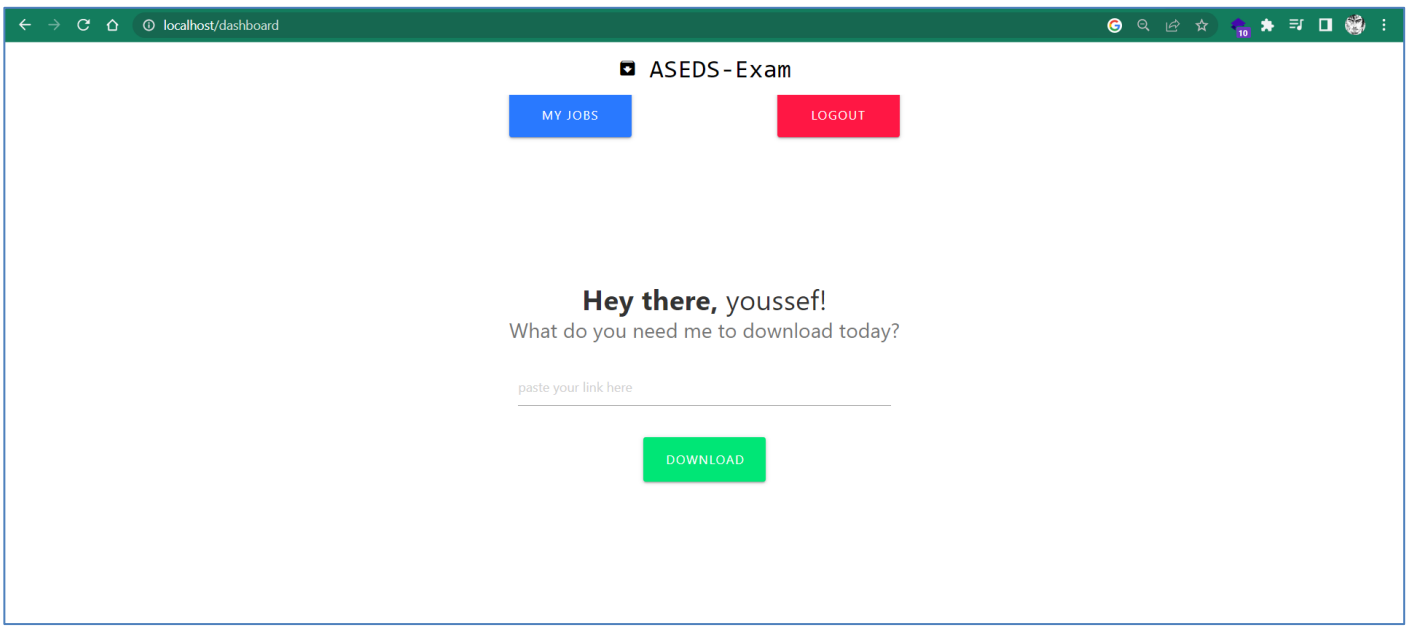
- Register Screen



- Sign in screen



- User home screen



Part Two

1)

Creation of Kubernetes manifest files:

a) Deployments:

- Mongo DB deployment:

mongodb-deployment.yaml U

k8s > mongodb-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb-service
  labels:
    app: mongodb-service
spec:
  replicas: 2
  selector:
    matchLabels:
      app: mongodb-service
  template:
    metadata:
      labels:
        app: mongodb-service
    spec:
      containers:
        - name: mongodb-service
          image: mongo
          ports:
            - containerPort: 27017
          volumeMounts:
            - name: mongodb-storage
              mountPath: /data/db
      volumes:
        - name: mongodb-storage
          persistentVolumeClaim:
            claimName: mongodb-persistent-volume-claim
```

- Backend deployment:

backend-deployment.yaml U

k8s > backend-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
  labels:
    app: backend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: backend
          image: youssefzahi/backend-img
          ports:
            - containerPort: 5000
```

- Frontend deployment:

frontend-deployment.yaml U

k8s > frontend-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  labels:
    app: frontend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend
          image: youssefzahi/frontend-image
          ports:
            - containerPort: 80
          envFrom:
            - configMapRef:
                name: frontend-configmap
```

b) Services:

- Mongodb

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mongodb-service
spec:
  type: ClusterIP
  selector:
    app: mongodb-service
  ports:
    - protocol: TCP
      port: 27017
      targetPort: 27017
```

- Backend:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend
spec:
  type: ClusterIP
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 5000
      targetPort: 5000
```

- Frontend:

frontend-cluster-ip-service.yaml U

k8s > frontend-cluster-ip-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend-cluster-ip-service
spec:
  type: ClusterIP
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

c) ConfigMaps:

- Frontend

frontend-configmap.yaml U

k8s > frontend-configmap.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: frontend-configmap
data:
  CLOUDL_SERVER: "localhost"
```

d) Persisted Volume Claim:

- Mongo DB

mongodb-persistent-volume-claim.yaml U

k8s > mongodb-persistent-volume-claim.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-persistent-volume-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

2)

Configure Liveness, Readiness and Startup Probes

- To make sure our application will be running successfully, we will be using Kubernetes health check probes:

```yaml
livenessProbe:
  httpGet:
    path: /healthCheck
    port: backend-port
  timeoutSeconds: 15
  periodSeconds: 120

readinessProbe:
  httpGet:
    port: backend-port
    path: /healthCheck
  timeoutSeconds: 15
  periodSeconds: 13

startupProbe:
  tcpSocket:
    port: backend-port
  failureThreshold: 10
  periodSeconds: 10
```

Figure: Backend deployment YAML file

```yaml
livenessProbe:
  httpGet:
    path: /healthCheck
    port: frontend-port
  timeoutSeconds: 15
  periodSeconds: 120

readinessProbe:
  httpGet:
    port: frontend-port
    path: /healthCheck
  timeoutSeconds: 15
  periodSeconds: 13

startupProbe:
  tcpSocket:
    port: frontend-port
  failureThreshold: 10
  periodSeconds: 10
```

Figure: Frontend deployment YAML file

```yaml
livenessProbe:
  tcpSocket:
      port: mongodb-port
  timeoutSeconds: 15
  periodSeconds: 120

startupProbe:
  tcpSocket:
    port: mongodb-port
  failureThreshold: 10
  periodSeconds: 10
```

Figure: Mongo DB deployment YAML file

- The liveness probes determine whether an application running in a container is in a healthy state. If the liveness probe detects an unhealthy state, then Kubernetes kills the container and tries to redeploy it.

- Readiness probes determine whether a container is ready to serve requests. If the readiness probe returns a failed state, then Kubernetes removes the IP address for the container from the endpoints of all Services.

- startup probe verifies whether the application within a container is started. Startup probes run before any other probe, and, unless it finishes successfully, disables other probes. If a container fails its startup probe, then the container is killed and follows the pod's restartPolicy.

    - The application will have a maximum of 100 seconds (10 * 10 = 100) to finish its startup.
    - The application will have a periodic liveness and readiness health check every 13 seconds.
- For the http Check, I have added new routes in backend and frontend:

```
// health check for kubernetes
app.get('/api/healthCheck', (req, res) => {
    res.status(200).send("The app is healthy");
});
```

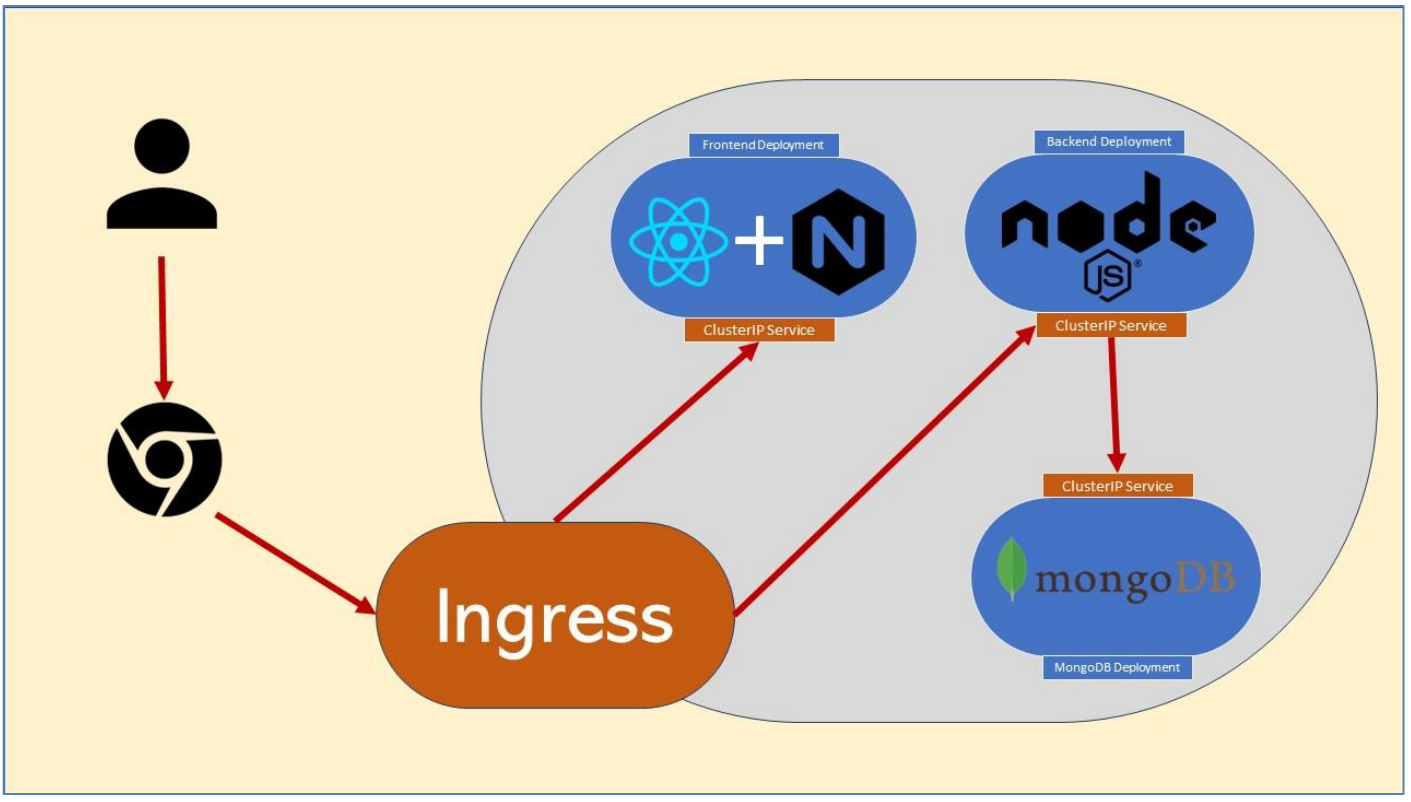
```
class App extends Component {
    render() {
        return (
            <Provider store={store}>
                <Router>
                    <div className="App">
                        <NavBar />
                        <Route exact path="/healthCheck" component={HealthCheck} />
                        <Route exact path="/" component={Home} />
                        <Route exact path="/register" component={Register} />
                        <Route exact path="/login" component={Login} />


                        <Switch>
                            <PrivateRoute exact path="/dashboard" component={Dashboard} />
                            <PrivateRoute exact path="/jobs" component={Jobs} />
                        </Switch>
                    </div>
                </Router>
            </Provider>
        );
    }
}
```

```
frontend > src > JS HealthCheck.js > ...
import React from 'react'

const HealthCheck = () => {
  return (
    <div>The app is healthy</div>
  )
}

export default HealthCheck
```

3)

Create and configure Ingress service

- Ingress Service manages external access to the services in a cluster. More specifically, we will be using the NGINX Ingress Controller.



- I have enabled NGINX Ingress Controller using this command:

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.5.1/deploy/static/provider/cloud/deploy.yaml
```

- And for the configuration file:

ingress-service.yaml M

k8s > ingress-service.yaml > ...

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-service
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/use-regex: "true"
spec:
  rules:
    - http:
        paths:
          - path: /?(.*)
            pathType: Prefix
            backend:
              service:
                name: frontend-cluster-ip-service
                port:
                  number: 80
          - path: /api/?(.*)
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 5000
```

4)

- After the deploying the application in Kubernetes and use an Ingress to get access to the routes from the cluster, The app worked successfully.

Links and References

- ***GitHub Repository:***
  - *Link: Visit Here*

- ***DockerHub Frontend Image:***
  - *Link: Visit Here*

- ***DockerHub Backend Image:***
  - *Link: Visit Here*

- ***Docker Multi-stage building:***
  - *Link: Reference*

- ***Docker best practices for writing a Dockerfile:***
  - *Link-1: Reference*
  - *Link-2: Reference*

- ***Kubernetes Health Check Probs:***
  - *Link-1: Reference*
  - *Link-2: Reference*
  - *Link-3: Reference*