Docker Setup

- **What is Docker?**
  Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.

- Open the terminal on your local system and run the following command:
  docker --version

- If Docker is not installed on your system, install it from the following link:
  https://www.docker.com/products/docker-desktop/

- Open the terminal on your local system and run the following command:
  docker --version
  Ex:- Docker version 27.1.1, build 6312585

- Create a DockerHub account using it from the following link:
  https://www.docker.com/products/docker-hub/

- Click on Create hub account and complete signup activity

- Set up the Hires project in the desired location.

- You can download the project using the following link:
  https://rr0z-my.sharepoint.com/:u:/g/personal/devlopment_codes_ohminsights_ca/ETmpYzc8NtZMiYEvzrIpalQBFnpAEWAd2QoFVeVVt-UoQA?e=GErnMW

HIRES Docker Frontend Setup(ReactJS)

- Open the terminal on your local system and run the following command:
  node -v

- If Node.js is not installed on your system, install it from the following link:
  https://nodejs.org/en

- Set up the Hires project in the desired location.

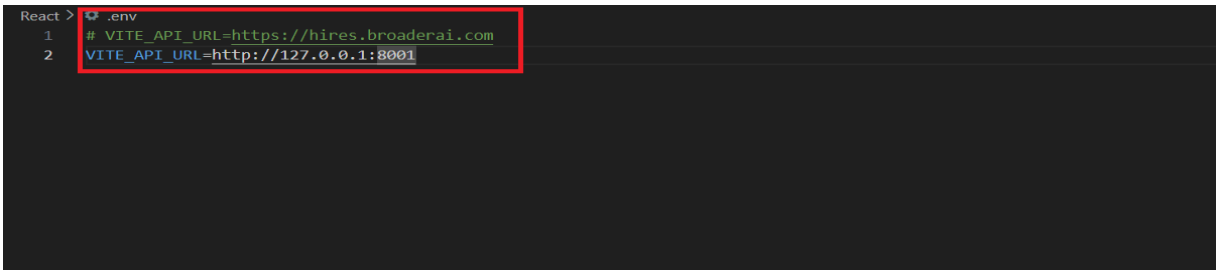
- Open the terminal in the React project directory and run the following command:
  code .

- This will open Visual Studio Code. In VS Code, open the terminal and run the following command:
  npm install

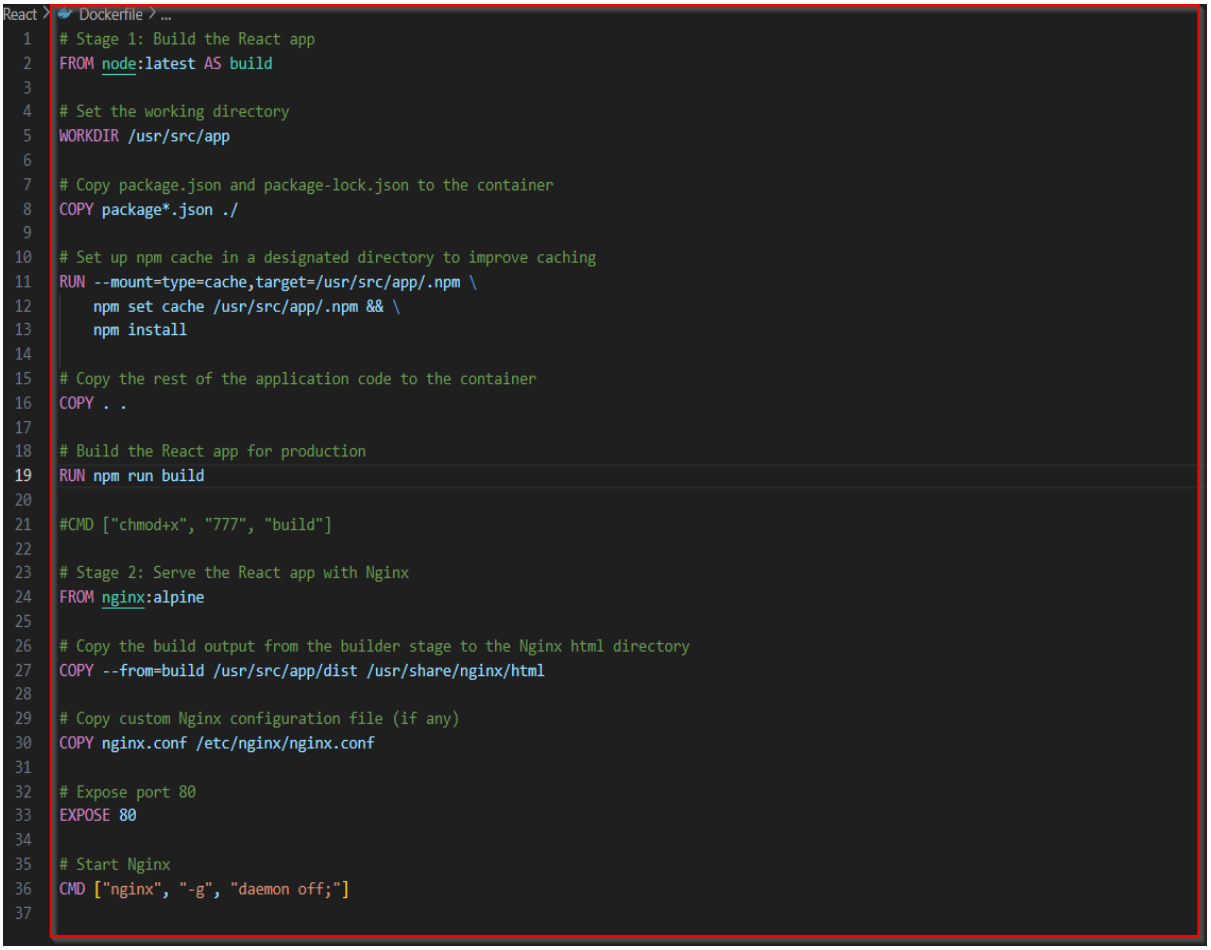
- After all dependencies have been installed, run the following command
  npm run dev

- Open the link provided in the browser
  Ex:- ➜ Local: http://localhost:5173/

- Create a .env file for to set api project base url:
  1. The backend (likely a Django application in your case) is expected to be running at the URL defined by `VITE_API_URL`. When the front-end sends a request to `http://127.0.0.1:8001`, the backend should be prepared to handle the request at that URL.If the front-end is running in a production environment, it will target `https://hires.broaderai.com`. If in development, it will target the local server running on `http://127.0.0.1:8001` .



- Create a Docker file for frontend project(reactJS):



Steps in Stage 1:

1. **Base Image**:
   - `FROM node:latest AS build`: Uses the latest Node.js image as the base. Node.js is necessary for building the React application.

2. **Working Directory**:
   - `WORKDIR /usr/src/app`: Sets the working directory inside the container to `/usr/src/app`. All subsequent commands are executed in this directory.

3. **Copy Dependencies**:
   - `COPY package*.json ./`: Copies `package.json` and `package-lock.json` into the container. These files define the dependencies for the React app.

4. **Install Dependencies**:
   - `RUN --mount=type=cache,target=/usr/src/app/.npm \`:
     - The `--mount=type=cache` option caches the npm modules in a designated directory to speed up the build process in future runs.
   - `npm install`: Installs the npm dependencies defined in `package.json`.

5. **Copy Application Code**:
   - `COPY . .`: Copies the entire project code (everything in the current directory) into the container.

6. **Build the App**:
   - `RUN npm run build`: Runs the build command to compile the React app into static files for production. These files will be stored in the `build` or `dist` directory (depending on your React configuration).

Steps in Stage 2:

1. **Base Image**:
   - `FROM nginx:alpine`: Uses a lightweight Nginx image based on Alpine Linux. Nginx is used to serve the static files generated by the React build.

2. **Copy Build Output**:
   - `COPY --from=build /usr/src/app/dist /usr/share/nginx/html`: Copies the compiled React app from the build stage (`/usr/src/app/dist`) to the Nginx default HTML directory (`/usr/share/nginx/html`). Nginx will serve the files from this directory.

3. **Nginx Configuration**:
    - `COPY nginx.conf /etc/nginx/nginx.conf`: Optionally copies a custom Nginx configuration file into the container. If you have specific Nginx settings, such as caching rules or redirects, this file would define them.

4. **Expose Port**:
    - `EXPOSE 80`: Exposes port 80, which is the default port Nginx listens on. This allows the container to serve the React app over HTTP.

5. **Start Nginx**:
    - `CMD ["nginx", "-g", "daemon off;"]`: Starts Nginx in the foreground, keeping the container running.

HIRES Backend Setup (Python-django)

Check Python Installation

- Open Command Prompt.

- Type the following command and press Enter:
  python –version

- If Python is installed, you will see the version number (e.g., Python 3.9.5). If not, you may get an error or a message indicating that the command is not recognized.
  - O https://www.python.org/ftp/python/3.12.5/python-3.12.5-amd64.exe

- Install PostgreSQL
  - O If PostgreSQL is not installed, download it from https://www.postgresql.org/

- Create a virtual environment using one of the following commands Make environment using below command
  Python –m venv env
  Or
  virtualenv env

- Activate environment
  source env/bin/activate
  Or
  .\env\Scripts\activate

- Open pgadmin and make database using project name.

- Configure Database in Django
  - O Open the settings.py file in your Django project.
  - O Set the database name, username, and password as follows:

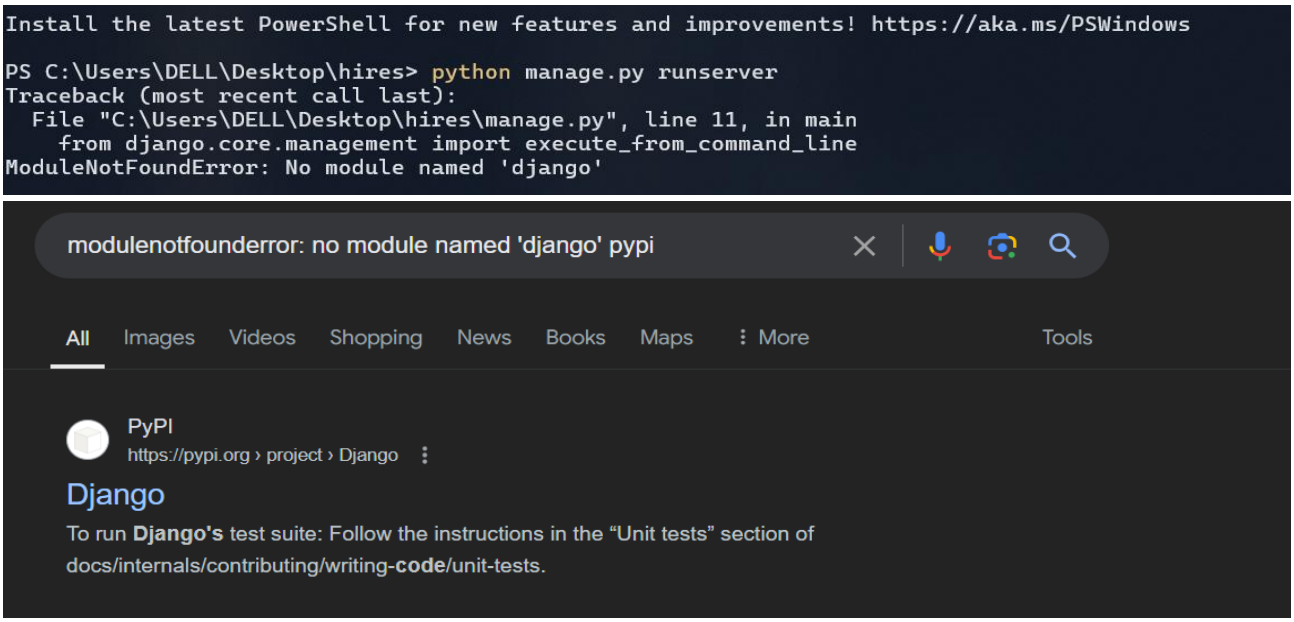
e.g

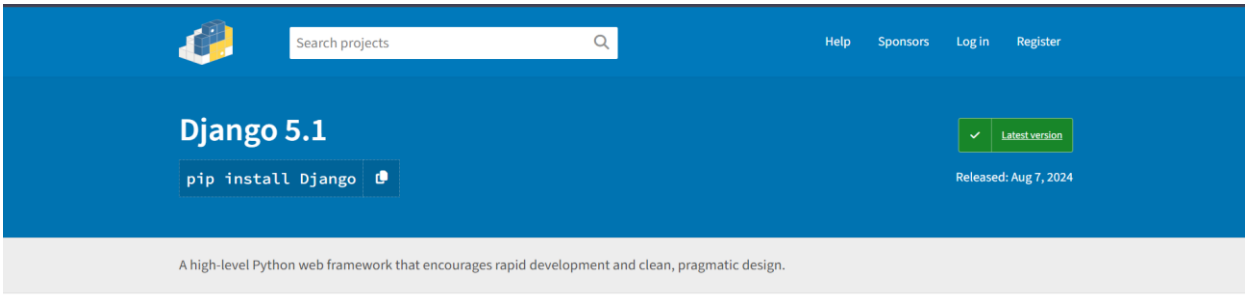
```
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.postgresql',
      'NAME': 'hires_db',
      'USER': 'postgres',
      'PASSWORD': 'sql@123',
      'HOST': 'db',
      'PORT': '5432',
   }}
```

- Install requirement using below code
  pip install –r requirement.txt

- Start the server using the following command. If there are no errors, you will get a link like http://127.0.0.1:8000 Python manage.py runserver

- Handling Module Errors:
  O If you encounter a ModuleNotFoundError, search for the module name on Google along with "pypi" to find the installation command. Install the missing module using:





- Create .env file for backend project:

These settings are used to configure Django to send emails using Gmail's SMTP server.

1. **`EMAIL_USE_TLS = True`**: Enables Transport Layer Security (TLS) for email communication, which ensures that the email content is encrypted during transmission.
2. **`EMAIL_HOST = 'smtp.gmail.com'`**: Specifies the SMTP server to be used for sending emails. In this case, it's Gmail's SMTP server.
3. **`EMAIL_HOST_USER = 'yashpp5545@gmail.com'`**: The email address that will be used to send the emails. This is your Gmail account.
4. **`EMAIL_HOST_PASSWORD = 'jyixqsmsdrbgyvsv'`**: The password or app-specific password for the Gmail account. This allows Django to authenticate with the Gmail SMTP server.
5. **`EMAIL_PORT = 587`**: The port number to be used when connecting to the SMTP server. Port 587 is commonly used for email submission with TLS.
6. **`SECRET_KEY`**: This is a crucial setting in Django projects. It's a random string used for cryptographic signing in Django, such as hashing passwords, generating tokens, and other security-related tasks.

- Create a entrypoint.sh for backend project:

Django > $ entrypoint.sh

```
#!/bin/bash

# entrypoint.sh

# Run Django migrations
echo "Running migrations..."
python manage.py makemigrations

python manage.py migrate

# Collect static files (if applicable)
echo "Collecting static files..."
python manage.py collectstatic --noinput

# Start Gunicorn
echo "Starting Server..."
exec python manage.py runserver 0.0.0.0:8001
```

1. **`python manage.py makemigrations`**: This command scans for any changes in your Django models and prepares migration files. These migration files tell Django how to modify the database schema to match the current state of your models.

2. **`python manage.py migrate`**: This command applies the migrations to your database, ensuring that your database schema is up-to-date with your models.

3. **`python manage.py collectstatic --noinput`**: This command gathers all static files (like CSS, JavaScript, and images) from your Django apps and places them into the directory specified by the `STATIC_ROOT` setting. This is especially important in production, where static files are served directly by the web server (e.g., Nginx) rather than Django.

4. The `--noinput` flag is used to prevent the command from prompting for any user input, which is useful in automated environments like Docker.

5. **`exec python manage.py runserver 0.0.0.0:8001`**: This command starts the Django development server, making the application accessible on all network interfaces (`0.0.0.0`) and listening on port `8001`.

6. The `exec` command replaces the current shell with the Django server process, ensuring that the server becomes the main process of the container.

- Create a Dockerfile for backend project:

```
Django > Dockerfile > ...
1   # backend/Dockerfile
2   FROM python:3.12
3
4   WORKDIR /hires
5
6   COPY requirements.txt requirements.txt
7   RUN pip install -r requirements.txt
8
9   COPY . .
10
11  # Copy the entrypoint script
12  COPY entrypoint.sh /entrypoint.sh
13  RUN chmod +x /entrypoint.sh
14
15  # Set the entrypoint to the script
16  ENTRYPOINT ["/entrypoint.sh"]
17
```

1. **`FROM python:3.12`**: This specifies the base image for your Docker container. In this case, it's an official Python 3.12 image. This image includes a Python interpreter and the necessary environment to run Python applications.

2. **`WORKDIR /hires`**: This sets the working directory inside the container to `/hires`. All subsequent commands will be run relative to this directory. It's a good practice to keep your project files organized in a specific directory within the container.

3. **`COPY requirements.txt requirements.txt`**: This copies the `requirements.txt` file from your local machine to the container. The `requirements.txt` file lists all the Python packages your project depends on.

4. **`RUN pip install -r requirements.txt`**: This installs all the dependencies listed in the `requirements.txt` file using `pip`. This ensures that your application has all the necessary libraries installed in the container.

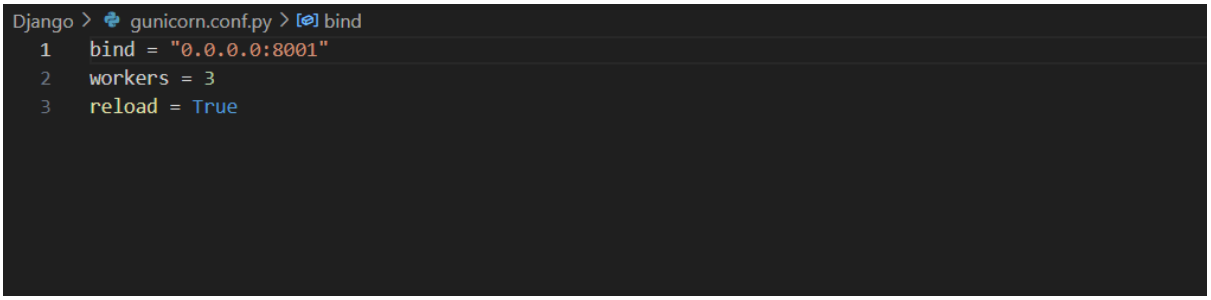
5. **`COPY . .`**: This copies all the files from your local project directory to the working directory (`/hires`) in the container. This includes your Django application code, configuration files, and any other necessary files.

6. **`COPY entrypoint.sh /entrypoint.sh`**: This copies the `entrypoint.sh` script into the root of the container's file system.

7. **`RUN chmod +x /entrypoint.sh`**: This makes the `entrypoint.sh` script executable. This is necessary because the script needs to be run when the container starts.

8. **`ENTRYPOINT ["/entrypoint.sh"]`**: This sets the `entrypoint.sh` script as the entrypoint for the container. When the container starts, this script will be executed. It handles tasks like running migrations, collecting static files, and starting the Django development server (as defined in your `entrypoint.sh` script).

- Create a gunicorn.conf. File for backend project:



1. **`bind = "0.0.0.0:8001"`**: This line specifies the address and port on which the server will listen. `0.0.0.0` means the server will accept connections on all available IP addresses of the host machine, and `8001` is the port number on which it will listen for incoming connections.

2. **`workers = 3`**: This sets the number of worker processes that Gunicorn will spawn to handle incoming requests. More workers can help handle more simultaneous requests by parallelizing the work, but the optimal number of workers depends on the hardware and the specific application.

3. **`reload = True`**: This enables automatic reloading of the server when code changes are detected. This is useful during development, as it allows you to see changes without restarting the server manually. However, it's generally not recommended to use this setting in a production environment due to potential performance overhead.

- Create a .env file for Hires Docker project:

```
.env
POSTGRES_USER=postgres
POSTGRES_PASSWORD=sql@123
POSTGRES_DB=hires_db

EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = 'yashpp5545@gmail.com'
EMAIL_HOST_PASSWORD = 'jyixqsmsdrbgyvsv'
EMAIL_PORT = 587


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/

SECRET_KEY = os.getenv('DJANGO_SECRET_KEY', 'django-insecure-default-key')
```

PostgreSQL Database Configuration

1. **`POSTGRES_USER=postgres`**: The username for accessing the PostgreSQL database. In this case, it's set to `postgres`, which is the default superuser.

2. **`POSTGRES_PASSWORD=sql@123`**: The password for the PostgreSQL user specified above.

3. **`POSTGRES_DB=hires_db`**: The name of the PostgreSQL database that Django will use.

Email Configuration

1. **`EMAIL_USE_TLS = True`**: This indicates that TLS (Transport Layer Security) should be used to encrypt the connection between Django and the email server.

2. **`EMAIL_HOST = 'smtp.gmail.com'`**: The SMTP server to use for sending email. In this case, it's Gmail's SMTP server.

3. **`EMAIL_HOST_USER = 'yashpp5545@gmail.com'`**: The email address that will be used to send emails.

4. **`EMAIL_HOST_PASSWORD = 'jyixqsmsdrbgyvsv'`**: The password for the email account specified above. (Be cautious with sharing such sensitive information.)

5. **`EMAIL_PORT = 587`**: The port used for the email server. Port 587 is commonly used for SMTP with TLS.

Django Secret Key

1. **`SECRET_KEY = os.getenv('DJANGO_SECRET_KEY', 'django-insecure-default-key')`**: The secret key used by Django for cryptographic signing. It's important for securing sessions and other cryptographic operations. It pulls the key from an environment variable named `DJANGO_SECRET_KEY`. If the environment variable is not set, it falls back to a default key (`'django-insecure-default-key'`), which is not secure and should be replaced with a unique key for production.

- Create a docker-compose.yml for docker hires project:

Docker-compose.yml

```yaml
version: '3.9'

services:
  db:
    image: postgres:latest
    volumes:
      - postgres_data:/var/lib/postgresql/data/
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    networks:
      - backend
    restart: always

  hires:
    build: ./Django
    volumes:
      - ./Django:/hires
    networks:
      - backend
    depends_on:
      - db
    ports:
      - 8001:8001
    #restart: always

  react-production:
    build: ./React
    volumes:
      - ./React:/usr/src/app
    networks:
      - frontend
    ports:
      - 80:80


volumes:
  postgres_data:

networks:
  backend:
  frontend:
```

`version: '3.9'`

Specifies the version of the Docker Compose file format being used. Version `3.9` is compatible with Docker Engine 19.03.0 and higher.

`Services`

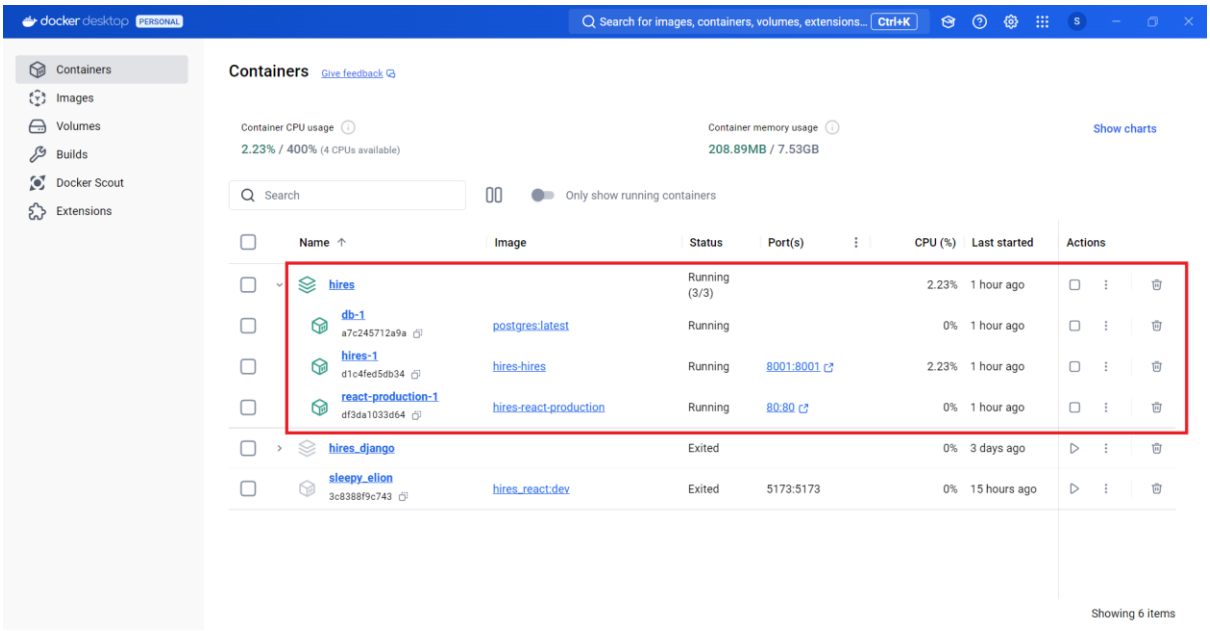
Defines the different services (containers) that make up the application. Each service runs in its own container.

1. **`db`**:
    - **`image: postgres:latest`**: Uses the latest version of the official PostgreSQL image from Docker Hub.
    - **`volumes`**:
        - **`postgres_data:/var/lib/postgresql/data/`**: Persists the database data across container restarts by mounting a named volume (`postgres_data`) to the PostgreSQL data directory.
    - **`environment`**:
        - **`POSTGRES_USER: ${POSTGRES_USER}`**: Sets the PostgreSQL user from an environment variable.
        - **`POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}`**: Sets the PostgreSQL password from an environment variable.
        - **`POSTGRES_DB: ${POSTGRES_DB}`**: Sets the name of the PostgreSQL database from an environment variable.
    - **`networks`**:
        - **`backend`**: Connects the container to the `backend` network.
    - **`restart: always`**: Ensures the container restarts automatically if it stops or the Docker daemon restarts.
2. **`hires`**:
    - **`build: ./Django`**: Builds the Django service container using the Dockerfile located in the `./Django` directory.
    - **`volumes`**:
        - **`./Django:/hires`**: Mounts the local `./Django` directory to `/hires` in the container. This allows for live code updates if the local code changes.
    - **`networks`**:
        - **`backend`**: Connects the container to the `backend` network.
    - **`depends_on`**:
        - **`db`**: Ensures the `hires` service starts only after the `db` service is running.
    - **`ports`**:
        - **`8001:8001`**: Maps port 8001 on the host to port 8001 in the container.
    - **`restart: always`** (commented out): If uncommented, it would restart the container automatically if it stops or the Docker daemon restarts.

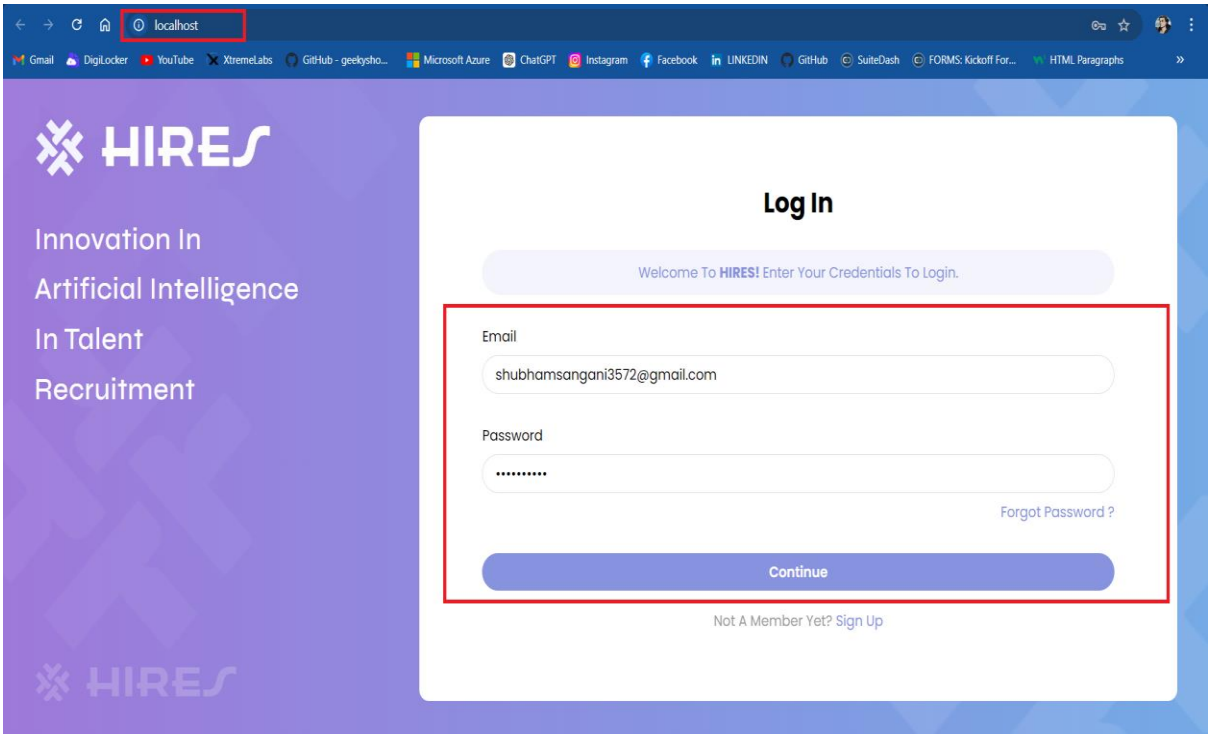
3. **`react-production`**:
    - **`build: ./React`**: Builds the React service container using the Dockerfile located in the `./React` directory.
    - **`volumes`**:
        - **`./React:/usr/src/app`**: Mounts the local `./React` directory to `/usr/src/app` in the container. Similar to the Django service, this allows for live updates to the React code.
    - **`networks`**:
        - **`frontend`**: Connects the container to the `frontend` network.
    - **`ports`**:
        - **`80:80`**: Maps port 80 on the host to port 80 in the container.

- This will open Visual Studio Code. In VS Code, open the terminal and run the following command for docker image and container:
  docker-compose up

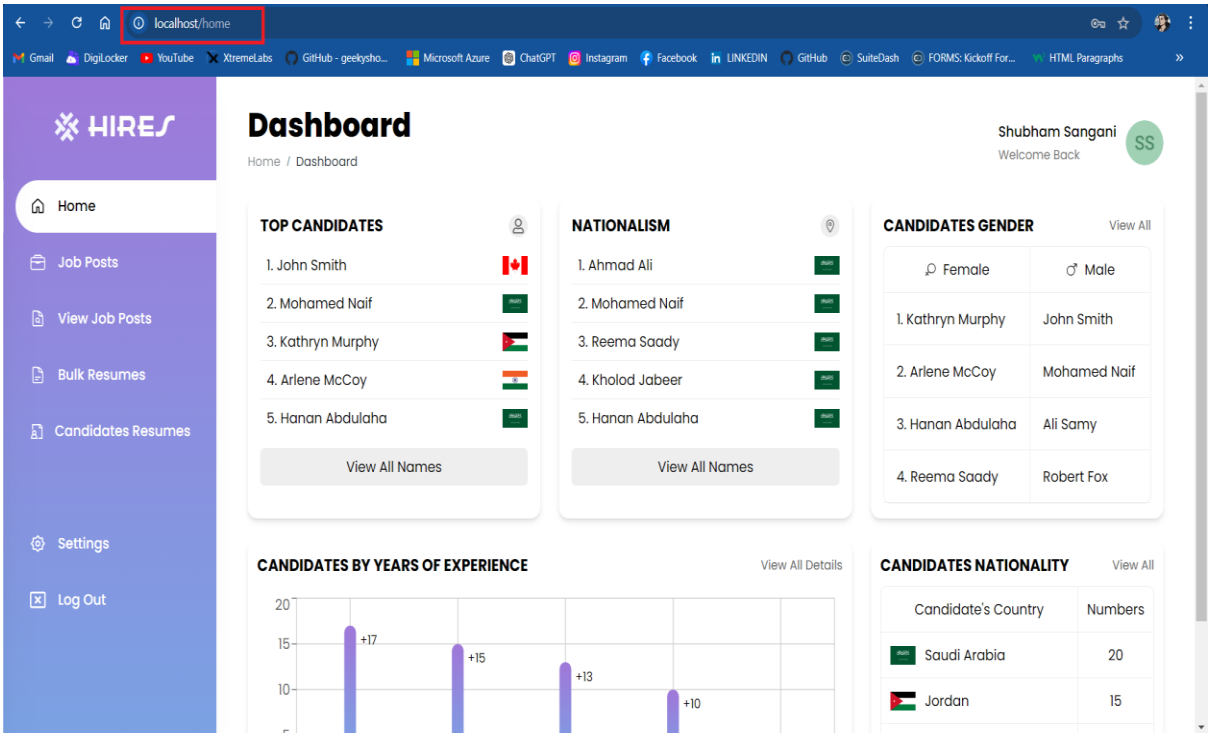
- Open Docker hub desktop, click on containers shows



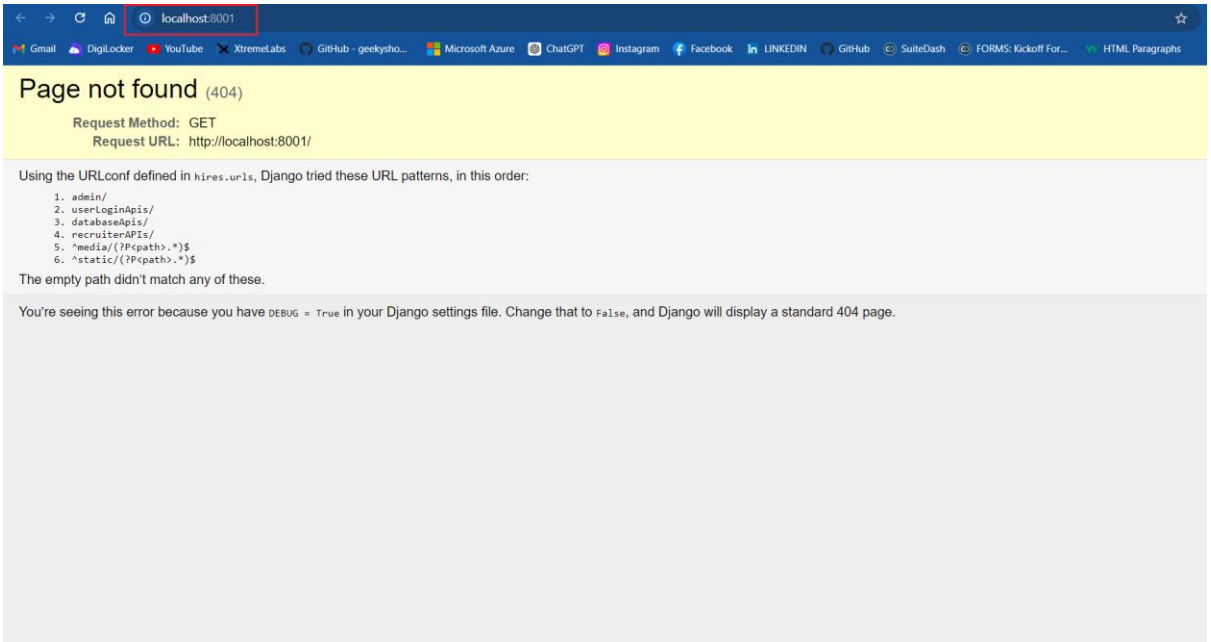
- Docker images hires-react-production > 80:80 (frontend project)
- Login page of hires-frontend-project



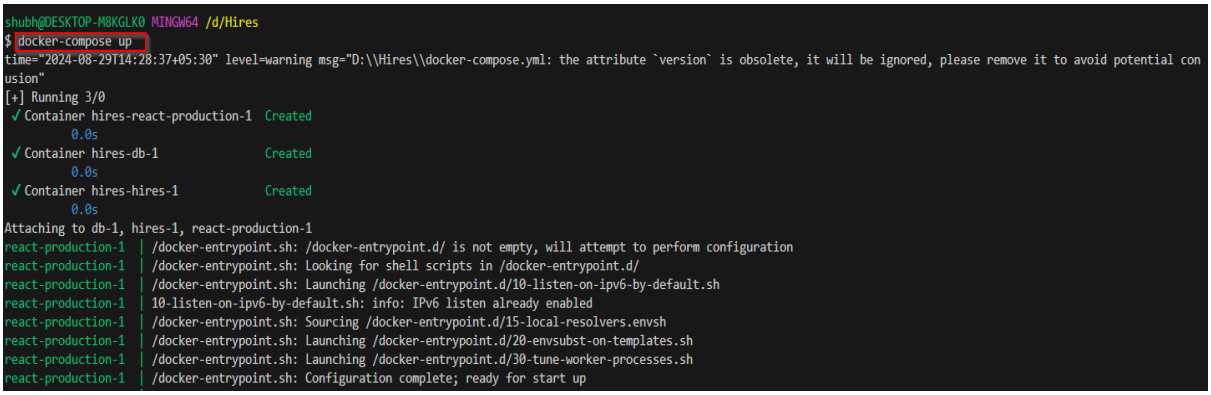
- Home page of hires-frontend-project



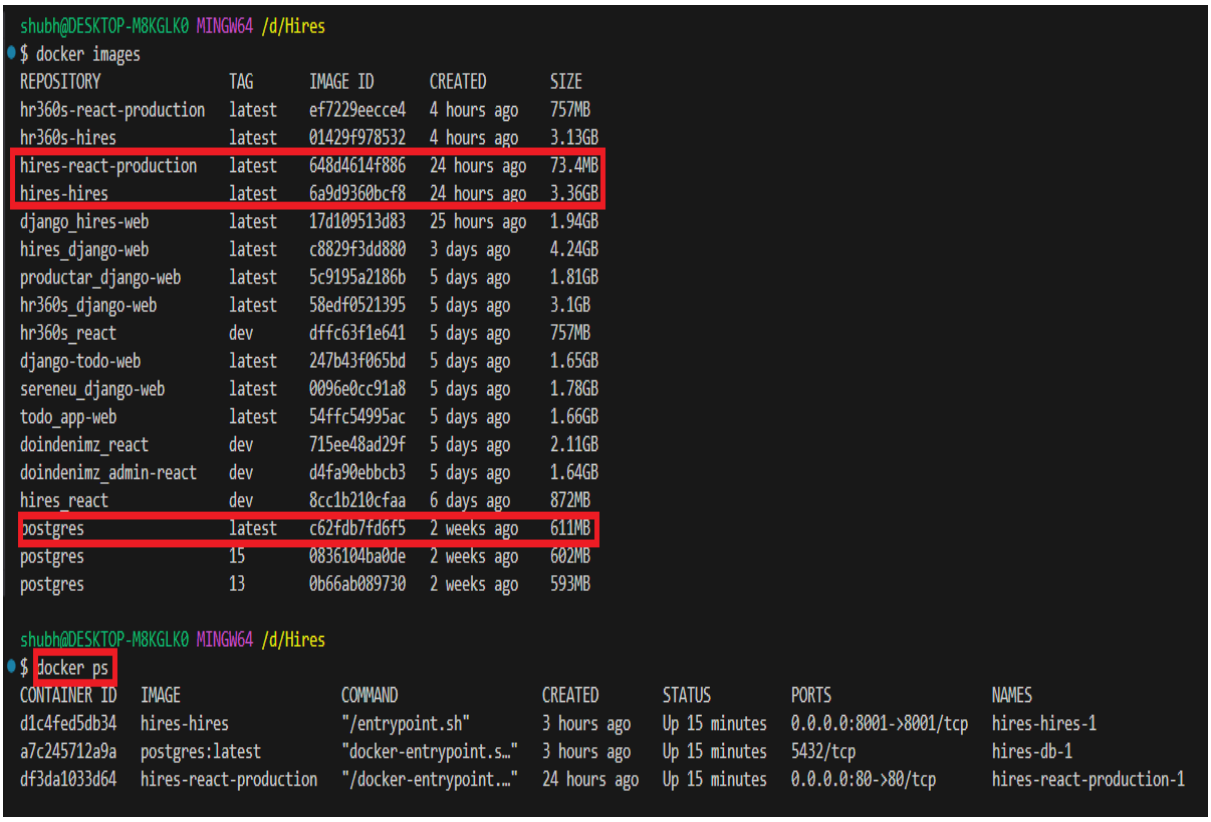
- Docker images hires-hires > 8001:8001 (backend project)



- This will open Visual Studio Code. In VS Code, open the terminal and run the following command for docker image and container:
  docker-compose up



- This will open Visual Studio Code. In VS Code, open the terminal and run the following command for docker images:
  docker images



- This will open Visual Studio Code. In VS Code, open the terminal and run the following command for docker running container:
  docker ps

```
shubh@DESKTOP-M8KGLK0 MINGW64 /d/Hires
$ docker images
REPOSITORY                TAG       IMAGE ID       CREATED          SIZE
hr360s-react-production   latest    ef7229eecce4   4 hours ago      757MB
hr360s-hires              latest    01429f978532   4 hours ago      3.13GB
hires-react-production    latest    648d4614f886   24 hours ago     73.4MB
hires-hires               latest    6a9d9360bcf8   24 hours ago     3.36GB
django_hires-web          latest    17d109513d83   25 hours ago     1.94GB
hires_django-web          latest    c8829f3dd880   3 days ago       4.24GB
productar_django-web      latest    5c9195a2186b   5 days ago       1.81GB
hr360s_django-web         latest    58edf0521395   5 days ago       3.1GB
hr360s_react              dev       dffc63f1e641   5 days ago       757MB
django-todo-web           latest    247b43f065bd   5 days ago       1.65GB
sereneu_django-web        latest    0096e0cc91a8   5 days ago       1.78GB
todo_app-web              latest    54ffc54995ac   5 days ago       1.66GB
doindenimz_react          dev       715ee48ad29f   5 days ago       2.11GB
doindenimz_admin-react    dev       d4fa90ebbcb3   5 days ago       1.64GB
hires_react               dev       8cc1b210cfaa   6 days ago       872MB
postgres                  latest    c62fdb7fd6f5   2 weeks ago      611MB
postgres                  15        0836104ba0de   2 weeks ago      602MB
postgres                  13        0b66ab089730   2 weeks ago      593MB

shubh@DESKTOP-M8KGLK0 MINGW64 /d/Hires
$ docker ps
CONTAINER ID   IMAGE                   COMMAND                  CREATED          STATUS          PORTS                    NAMES
d1c4fed5db34   hires-hires             "/entrypoint.sh"         3 hours ago      Up 15 minutes   0.0.0.0:8001->8001/tcp   hires-hires-1
a7c245712a9a   postgres:latest         "docker-entrypoint.s…"   3 hours ago      Up 15 minutes   5432/tcp                 hires-db-1
df3da1033d64   hires-react-production   "/docker-entrypoint.…"   24 hours ago     Up 15 minutes   0.0.0.0:80->80/tcp       hires-react-production-1
```

- This will open Visual Studio Code. In VS Code, open the terminal and run the following command for docker running container:
- Go to docker container for backend project shell with following command:
  docker exec -it <container-id> bash

Eg: **docker exec -it d1c4fed5db34 bash**

```
shubh@DESKTOP-M8KGLK0 MINGW64 /d/Hires
$ docker ps
CONTAINER ID   IMAGE                   COMMAND                  CREATED          STATUS          PORTS                    NAMES
d1c4fed5db34   hires-hires             "/entrypoint.sh"         3 hours ago      Up 17 minutes   0.0.0.0:8001->8001/tcp   hires-hires-1
a7c245712a9a   postgres:latest         "docker-entrypoint.s…"   3 hours ago      Up 17 minutes   5432/tcp                 hires-db-1
df3da1033d64   hires-react-production   "/docker-entrypoint.…"   24 hours ago     Up 17 minutes   0.0.0.0:80->80/tcp       hires-react-production-1

shubh@DESKTOP-M8KGLK0 MINGW64 /d/Hires
$ docker exec -it d1c4fed5db34 bash
root@d1c4fed5db34:/hires# ls
Dockerfile  databaseAPI  db.sqlite3  entrypoint.sh  env  gunicorn.conf.py  hires  manage.py  media  recruiterAPI  requirements.txt  static  staticfiles  templates  userloginAPI
root@d1c4fed5db34:/hires# exit
exit
```

- Go to docker container for frontend project shell with following command:
  docker exec -it <container-id> sh
  Eg: docker exec -it df3da1033d64 sh