Docker setup practice with django backend and react js frontend to learn about multiple containers :-

create the project django backend:-

Run this command in your terminal:

```
django-admin startproject backend
```

Navigate into the project folder

```
cd backend
```

Create an app)

Inside your project:

```
python manage.py startapp api
```

Install the packages

Run this in your terminal inside your project:

```
pip install djangorestframework django-cors-headers
```

backend/settings.py file :-

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'corsheaders',
    'api',
]
```

```
MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',
]
```

```
CORS_ALLOWED_ORIGINS = [
    "http://localhost:5173",
]
```

api/models.py file :-

```
from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=100)
    city = models.CharField(max_length=100)

    def __str__(self):
        return self.name
```

Create migrations (generate migration files):-

```
cd backend

python manage.py makemigrations
```

Apply migrations to database

```
python manage.py migrate
```

api/serializers.py file:-

```python
from rest_framework import serializers
from .models import Person

class PersonSerializer(serializers.ModelSerializer):
    class Meta:
        model = Person
        fields = '__all__'
```

api/views.py file code:-

```python
from rest_framework import viewsets
from .models import Person
from .serializers import PersonSerializer

class PersonViewSet(viewsets.ModelViewSet):
    queryset = Person.objects.all()
    serializer_class = PersonSerializer
```

api/urls.py file code:-

```python
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import PersonViewSet
```

```
router = DefaultRouter()
router.register(r'persons', PersonViewSet)

urlpatterns = [
    path('', include(router.urls)),
]
```

backend/urls.py file :-

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('api.urls')),
]
```

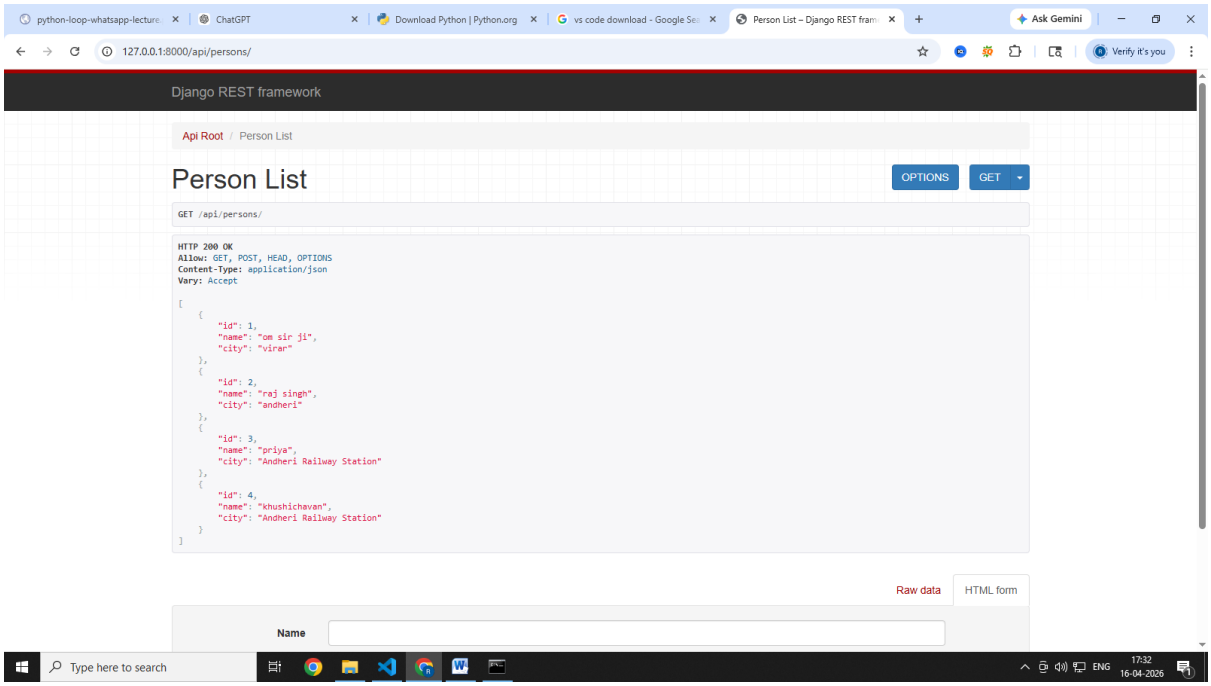
Then run it:-

cd backend

python manage.py runserver

and visit in browser

http://127.0.0.1:8000/api/persons/



Create React project with Vite

Run this command:

```
npm create vite@latest frontend
```

It will ask you:

- Framework → **React**
- Variant → **JavaScript** or **TypeScript**

---

3. Go into project folder

```
cd frontend
```

Install Axios

Run this inside your project folder (`frontend`):

```
npm install axios
```

`src/api.js`

```
import axios from "axios";

const API = axios.create({
  baseURL: "http://localhost:8000/api/"
});

export const getPersons = () => API.get("persons/");
export const addPerson = (data) => API.post("persons/", data);
export const updatePerson = (id, data) => API.put(`persons/${id}/`, data);
export const deletePerson = (id) => API.delete(`persons/${id}/`);
```

src/App.jsx file:-

```
import { useEffect, useState } from "react";
import { getPersons, addPerson, updatePerson, deletePerson } from "./api";

function App() {
  const [persons, setPersons] = useState([]);
  const [form, setForm] = useState({ name: "", city: "" });
  const [editId, setEditId] = useState(null);

  const fetchData = async () => {
    const res = await getPersons();
    setPersons(res.data);
  };

  useEffect(() => {
    fetchData();
  }, []);

  const handleSubmit = async (e) => {
    e.preventDefault();
    if (editId) {
      await updatePerson(editId, form);
      setEditId(null);
    } else {
      await addPerson(form);
    }
    setForm({ name: "", city: "" });
```

```
    fetchData();
  };

  const handleEdit = (person) => {
    setForm({ name: person.name, city: person.city });
    setEditId(person.id);
  };

  const handleDelete = async (id) => {
    await deletePerson(id);
    fetchData();
  };

  return (
    <div style={{ padding: "20px" }}>
      <h2>CRUD App</h2>

      <form onSubmit={handleSubmit}>
        <input
          placeholder="Name"
          value={form.name}
          onChange={(e) => setForm({ ...form, name: e.target.value })}
        />
        <input
          placeholder="City"
          value={form.city}
          onChange={(e) => setForm({ ...form, city: e.target.value })}
        />
        <button type="submit">
          {editId ? "Update" : "Add"}
        </button>
      </form>

      <ul>
        {persons.map((p) => (
          <li key={p.id}>
            {p.name} - {p.city}
            <button onClick={() => handleEdit(p)}>Edit</button>
            <button onClick={() => handleDelete(p.id)}>Delete</button>
          </li>
        ))}
      </ul>
    </div>
  );
}

export default App;
```

Start development server (run frontend):-

cd frontend

```
npm run dev
```

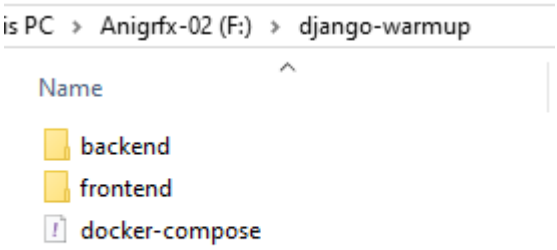
output:-

visit in browser

http://localhost:5173/

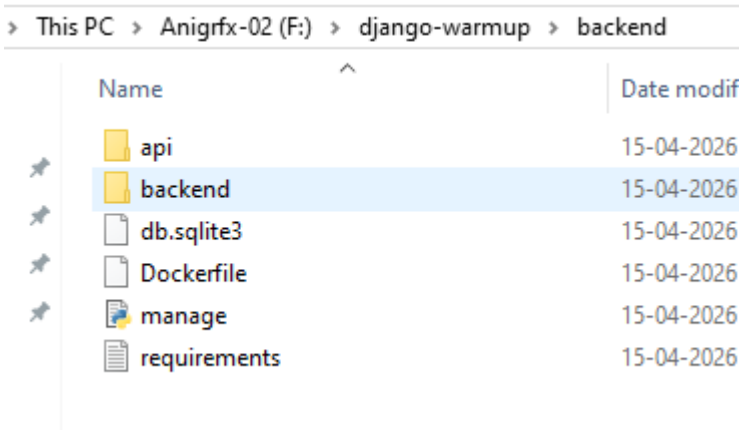
“Full Stack CRUD App using Django, React, Vite & Docker Compose”

Create root folder djago-warmup :-



Backend folder Dockerfile :-

```
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

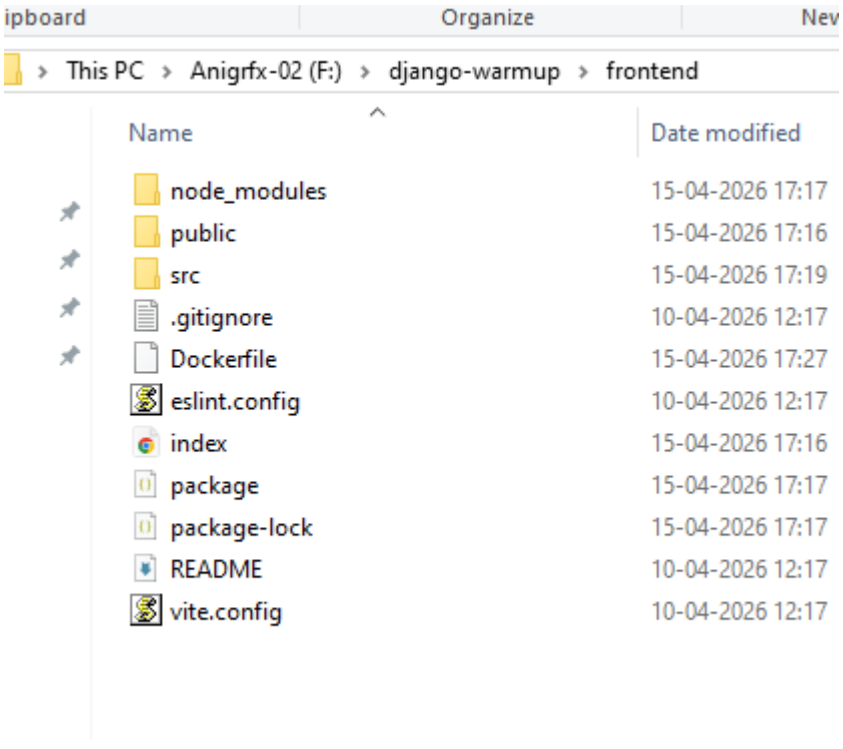


Frontend folder Dockerfile

```
FROM node:20

WORKDIR /app

COPY package.json package-lock.json ./

RUN npm install

COPY . .

EXPOSE 5173

CMD ["npm", "run", "dev", "--", "--host"]
```

docker-compose.yml | Dockerfile | docker-compose.yml | requirements.txt | Dockerfile

```
1  FROM node:20
2
3  WORKDIR /app
4
5  COPY package.json package-lock.json ./
6
7  RUN npm install
8
9  COPY . .
10
11 EXPOSE 5173
12
13 CMD ["npm", "run", "dev", "--", "--host"]
```



Docker Compose (Core Concept)

Explain:

 “Instead of running manually, we connect everything”

```
services:
  backend:
  frontend:
```

docker-compose.yml file inside root folder django-warmup:-

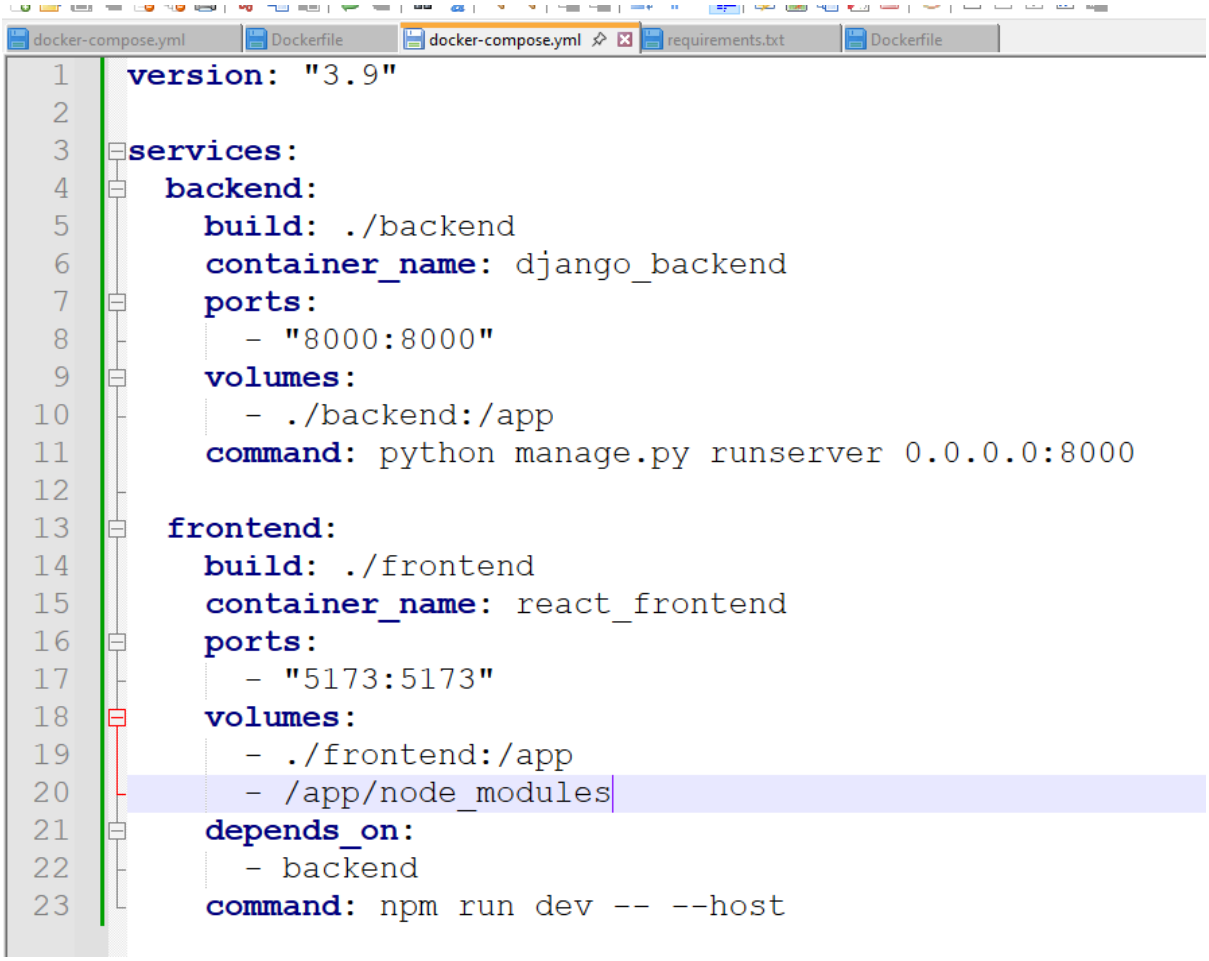
```
version: "3.9"

services:
  backend:
    build: ./backend
    container_name: django_backend
    ports:
      - "8000:8000"
    volumes:
      - ./backend:/app
    command: python manage.py runserver 0.0.0.0:8000

  frontend:
    build: ./frontend
    container_name: react_frontend
    ports:
      - "5173:5173"
    volumes:
      - ./frontend:/app
```

- /app/node_modules

depends_on:

- backend

command: npm run dev -- --host

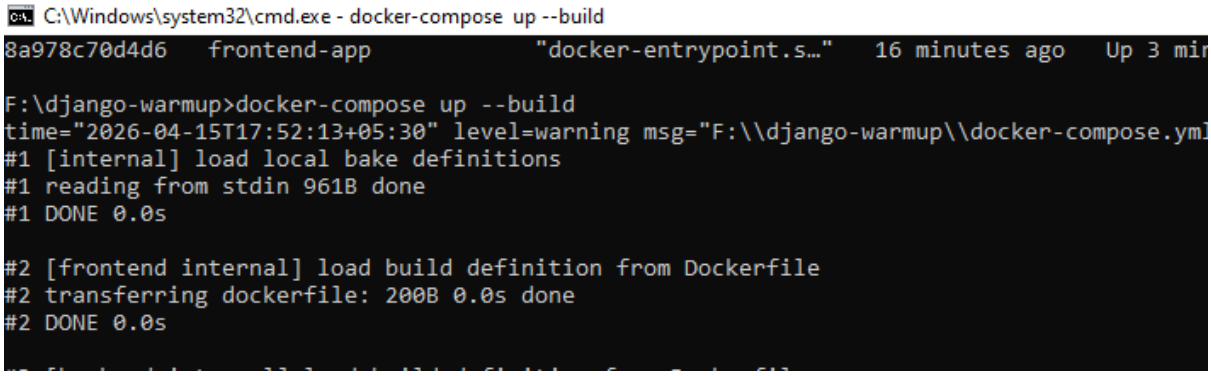


□ STEP 8: Run Everything Together

```
docker-compose up --build
```

Explain:

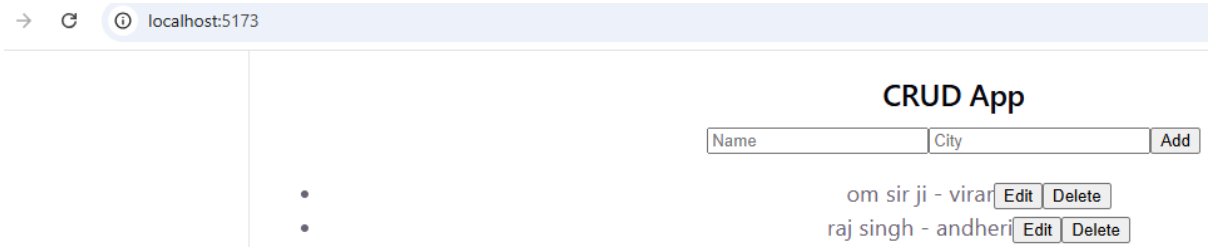
- one command starts full stack



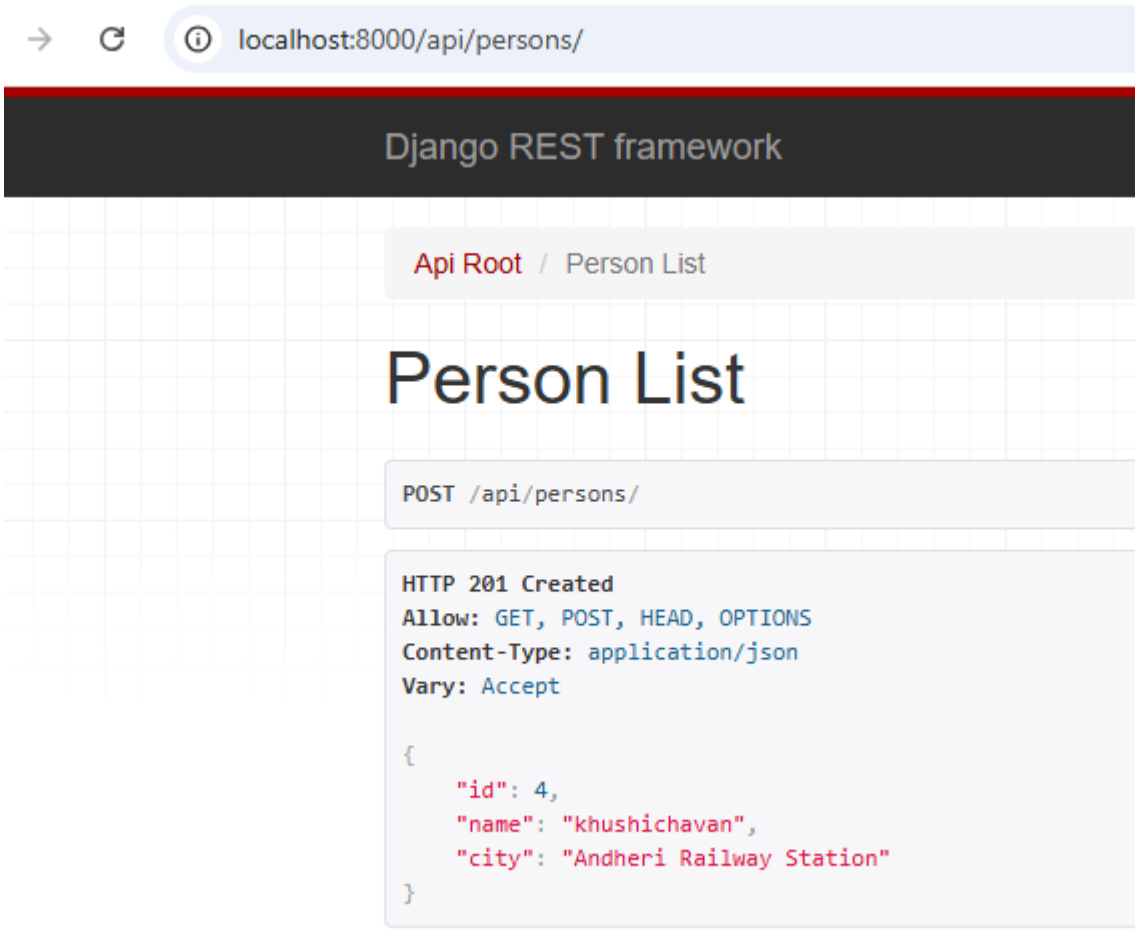
□ STEP 9: Final Testing

Open:

`Frontend:` http://localhost:5173



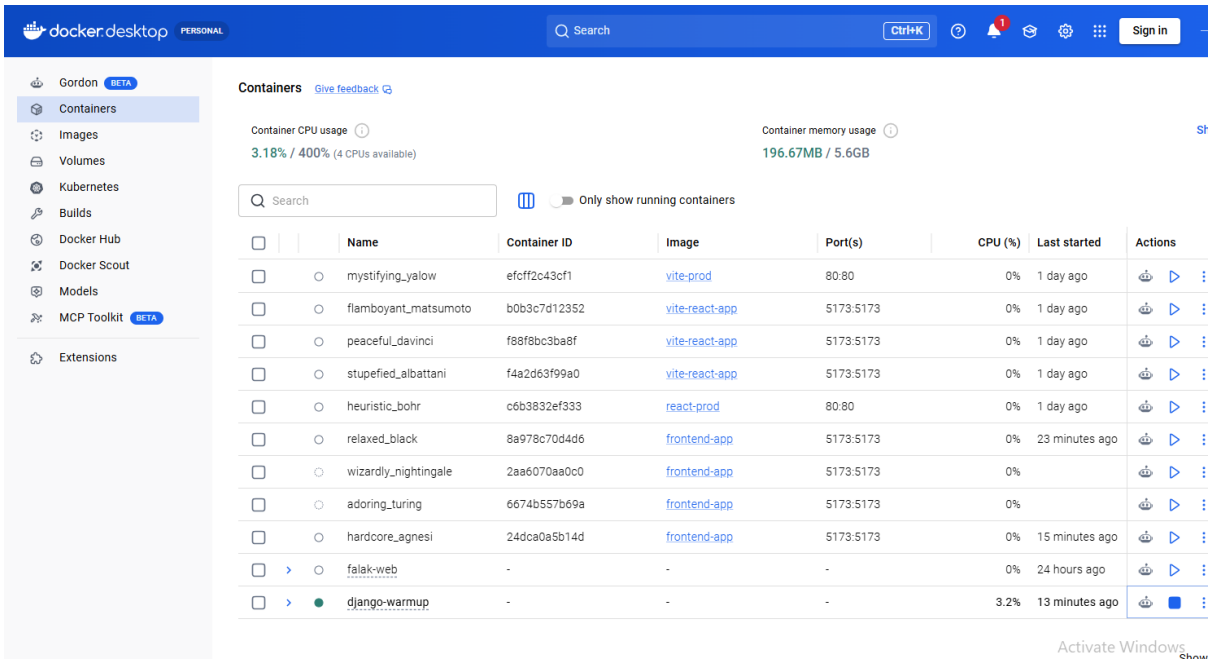
`Backend:` http://localhost:8000/api/persons/



Do live CRUD demo:

- Add user
- Edit user
- Delete user

And in your docker desktop you will see running containers :-



Great — now you want to move from **development setup → production-ready full stack**. This is the real industry step

We'll convert your setup:

- Backend → **Django**
- Frontend → **React + Vite**
- Orchestration → **Docker + Docker Compose**

PRODUCTION READY SETUP (STEP-BY-STEP)

---

1. KEY DIFFERENCE (IMPORTANT)

Dev Mode	Production Mode
`npm run dev`	`npm run build`
Django runserver	Gunicorn
Hot reload	Static files
Vite dev server	Built HTML/CSS/JS

---

2. FRONTEND (PRODUCTION BUILD)

Step 1: Build React app

Inside frontend:

```
npm run build
```

 This creates:

```
dist/
```

Step 2: Frontend Dockerfile (PRODUCTION)

```
FROM node:20 as build

WORKDIR /app
```

```
COPY package.json ./
RUN npm install

COPY . .

RUN npm run build


FROM nginx:alpine

COPY --from=build /app/dist /usr/share/nginx/html

EXPOSE 80

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

□ 3. BACKEND (PRODUCTION)

Step 1: Install production server

```
pip install gunicorn
```

Step 2: Backend Dockerfile

```
FROM python:3.11

WORKDIR /app

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["gunicorn", "backend.wsgi:application", "--bind", "0.0.0.0:8000"]
```

□ 4. PRODUCTION docker-compose.yml

```
version: "3.9"

services:
  backend:
    build: ./backend
    ports:
      - "8000:8000"

  frontend:
    build: ./frontend
    ports:
```

```
      - "80:80"
    depends_on:
      - backend
```

□ 5. FRONTEND API CHANGE (IMPORTANT)

Now backend is stable:

```
baseURL: "http://localhost:8000/api/"
```

OR production domain later:

```
baseURL: "https://yourdomain.com/api/"
```

□ 6. DJANGO PRODUCTION SETTINGS

In `settings.py`:

```
DEBUG = False

ALLOWED_HOSTS = ["*"]
```

□ 7. RUN PRODUCTION BUILD

```
docker-compose up --build
```

□ 8. FINAL OUTPUT

Service	URL
Frontend	http://localhost
Backend API	http://localhost:8000/api/

□ 9. WHAT YOU JUST BUILT

You upgraded from:

□ Dev system
✔ Production system

Now you have:

✔ Nginx frontend
✔ Gunicorn backend
✔ Docker production build
✔ Scalable architecture

□ BONUS (REAL INDUSTRY NEXT STEP)

If you want FULL professional level:

I can help you add:

- PostgreSQL database
- NGINX reverse proxy
- HTTPS (SSL)
- Deployment (AWS / DigitalOcean / Render)
- CI/CD pipeline (GitHub Actions)

□ FINAL SUMMARY

Production =

✔ Build frontend (`npm run build`)
✔ Use Nginx
✔ Use Gunicorn
✔ Disable dev servers
✔ Use Docker Compose

OPTION 1 (RECOMMENDED): Upload FULL PROJECT (BEST PRACTICE)

✅ Yes, you upload BOTH:

- backend folder (with Dockerfile)
- frontend folder (with Dockerfile)
- docker-compose.yml

📁 Final structure you upload

```
project/
│
├── backend/
│   ├── Dockerfile
│   ├── requirements.txt
│   └── manage.py
│
├── frontend/
│   ├── Dockerfile
│   ├── package.json
│   └── src/
│
└── docker-compose.yml
```

🚀 Then on server:

```
git clone your-repo
cd project
docker-compose up --build -d
```

✅ That's it.

🔥 WHY THIS WORKS

Docker Compose will:

✔ build backend image
✔ build frontend image
✔ connect both
✔ run everything automatically

🔥 OPTION 2: ONLY DOCKER IMAGES (ADVANCED)

Instead of uploading code:

1. Build images locally:

```
docker build -t backend ./backend
docker build -t frontend ./frontend
```

2. Push to Docker Hub:

```
docker push yourname/backend
docker push yourname/frontend
```

3. Server pulls images:

```
docker-compose up
```

□ WHAT YOU SHOULD NOT DO

□ Upload only frontend
□ Upload only backend
□ Run without docker-compose on server
□ Forget Dockerfiles

? SIMPLE ANSWER (IMPORTANT)

□ YES — you upload **both frontend + backend folders + docker-compose.yml**

? FINAL DEPLOY FLOW

```
GitHub / Server
   ↓
Full project (frontend + backend + compose)
   ↓
docker-compose up --build
   ↓
App goes live □
```

? BEST PRACTICE (REAL INDUSTRY)

Always deploy:

✔ full project
✔ docker-compose
✔ one command start