# **Pasos para levantar el proyecto fullstack con Docker**

## **1. Crear directorio raíz**

mkdir ecommerce && cd ecommerce

## **2. Crear docker-compose.yml y .env**

type nul > docker-compose.yml

type nul > .env

### **docker-compose.yml**

version: '3.9'

services:

backend:

build: ./backend

container\_name: django\_backend

command: *>*

sh -c "python manage.py makemigrations products &&

python manage.py migrate &&

python manage.py runserver 0.0.0.0:8000"

volumes:

- ./backend:/app

ports:

- "8080:8000"

env\_file:

- .env

depends\_on:

- db

db:

image: postgres:15

container\_name: postgres\_db

volumes:

- postgres\_data:/var/lib/postgresql/data

environment:

POSTGRES\_DB: ${POSTGRES\_DB}

POSTGRES\_USER: ${POSTGRES\_USER}

POSTGRES\_PASSWORD: ${POSTGRES\_PASSWORD}

ports:

- "5444:5432"

frontend:

build: ./frontend

container\_name: react\_frontend

volumes:

- ./frontend:/app

- /app/node\_modules

ports:

- "3030:3000"

stdin\_open: true

tty: true

command: npm start -- --host 0.0.0.0

volumes:

postgres\_data:

### **.env**

POSTGRES\_DB=ecommercedb

POSTGRES\_USER=ecommerceuser

POSTGRES\_PASSWORD=sena

POSTGRES\_HOST=postgres\_db

POSTGRES\_PORT=5432

## **3. Crear backend (Django + Clean Architecture)**

mkdir backend && cd backend

### **3.1 Crear Dockerfile dentro del directorio backend**

FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

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

COPY . .

EXPOSE 8000

### **3.2 Crear requirements.txt**

Django==5.1

djangorestframework==3.15

psycopg2-binary==2.9.9

python-decouple

django-cors-headers

### **3.3 Inicializar proyecto Django. Esto va en una sola linea.**

docker run --rm -v "C:\ecommerce\backend":/app -w /app python:3.11-slim bash -c "pip install django djangorestframework psycopg2-binary python-decouple django-cors-headers && django-admin startproject core ."

## **4. Crear Clean Architecture para products**

mkdir products

mkdir products\domain

mkdir products\application

mkdir products\infrastructure

mkdir products\interfaces

👉 Copias los archivos:

en domain/entities.py

from dataclasses import dataclass

from decimal import Decimal

@dataclass

class ProductEntity:

reference: str

name: str

price: Decimal

**En application/**[**services.py**](http://services.py)

from products.domain.entities import ProductEntity

class ProductService:

def \_\_init\_\_(self, repository):

self.repo = repository

def create(self, data):

entity = ProductEntity(\*\*data)

return self.repo.create(entity)

def list(self):

return self.repo.list()

def get(self, pk):

return self.repo.get(pk)

def update(self, pk, data):

entity = ProductEntity(\*\*data)

return self.repo.update(pk, entity)

def delete(self, pk):

return self.repo.delete(pk)

**En infraestructure/**[**models.py**](http://models.py)

from django.db import models

class ProductModel(models.Model):

reference = models.CharField(max\_length=50, unique=True)

name = models.CharField(max\_length=255)

price = models.DecimalField(max\_digits=12, decimal\_places=2)

**En instraestructure/**[**repositories.py**](http://repositories.py)

from .models import ProductModel

class DjangoORMProductRepository:

def create(self, entity):

return ProductModel.objects.create(\*\*entity.\_\_dict\_\_)

def list(self):

return ProductModel.objects.all()

def get(self, pk):

return ProductModel.objects.get(pk=pk)

def update(self, pk, entity):

obj = self.get(pk)

obj.reference = entity.reference

obj.name = entity.name

obj.price = entity.price

obj.save()

return obj

def delete(self, pk):

self.get(pk).delete()

**En interfaces/**[**serializers.py**](http://serializers.py)

from rest\_framework import serializers

from products.infrastructure.models import ProductModel

class ProductSerializer(serializers.ModelSerializer):

class Meta:

model = ProductModel

fields = ['id','reference','name','price']

**En interfaces/**[**views.py**](http://views.py)

from rest\_framework import viewsets, status

from rest\_framework.response import Response

from django.shortcuts import get\_object\_or\_404

from products.interfaces.serializers import ProductSerializer

from products.infrastructure.respositories import DjangoORMProductRepository

from products.application.services import ProductService

repo = DjangoORMProductRepository()

service = ProductService(repo)

class ProductViewSet(viewsets.ViewSet):

def list(self, request):

serializer = ProductSerializer(service.list(), many=True)

return Response(serializer.data)

def create(self, request):

serializer = ProductSerializer(data=request.data)

serializer.is\_valid(raise\_exception=True)

obj = service.create(serializer.validated\_data)

return Response(ProductSerializer(obj).data, status=status.HTTP\_201\_CREATED)

def retrieve(self, request, pk=None):

obj = get\_object\_or\_404(repo.list(), pk=pk)

return Response(ProductSerializer(obj).data)

def update(self, request, pk=None):

obj = repo.get(pk)

serializer = ProductSerializer(obj, data=request.data)

serializer.is\_valid(raise\_exception=True)

updated\_obj = service.update(pk, serializer.validated\_data)

return Response(ProductSerializer(updated\_obj).data)

def destroy(self, request, pk=None):

service.delete(pk)

return Response(status=status.HTTP\_204\_NO\_CONTENT)

En interfaces/[urls.py](http://urls.py)

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

from .views import ProductViewSet

router = DefaultRouter()

router.register(r'products', ProductViewSet, basename='product')

urlpatterns = [

path('', include(router.urls)),

]

Y en core/[settings.py](http://settings.py): Agregar

# Apps externas

'rest\_framework',

# Tu app de productos (solo la capa infraestructura)

'products',

* Configura DATABASES con PostgreSQL (usa las variables de .env)

**para el cors en** [settings.py](http://settings.py)

INSTALLED\_APPS += ['corsheaders']

MIDDLEWARE = [

'corsheaders.middleware.CorsMiddleware',

# ... tus otros middlewares

]

CORS\_ALLOW\_ALL\_ORIGINS = True # solo para desarrollo

instalar decouple para leer variables de entorno. Ya se instaló en un paso anterior.

from decouple import config

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': config('POSTGRES\_DB'),

'USER': config('POSTGRES\_USER'),

'PASSWORD': config('POSTGRES\_PASSWORD'),

'HOST': config('POSTGRES\_HOST'),

'PORT': config('POSTGRES\_PORT', default='5432'),

}

}

En core/urls.py:

path("api/", include("products.interfaces.urls")),

queda:

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path("api/", include("products.interfaces.urls")),

]

## 

## 

## **5. Crear frontend (CoreUI React)**

Desde raíz del proyecto:

git clone https://github.com/coreui/coreui-free-react-admin-template.git frontend

Cambiar al directorio del **frontend** e instalar Instalar dependencias adicionales.

npm install axios react-toastify jspdf

### **5.1 Crear Dockerfile en frontend**

FROM node:20

WORKDIR /app

COPY package.json package-lock.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD ["npm", "run", "start", "--", "--host", "0.0.0.0"]

### **5.2 Crear vista de productos**

mkdir frontend\src\views\products

cd frontend\src\views\products

type nul > [products.js](http://products.js)

En Products.js

import React, { useEffect, useState } from "react";

import {

CCard, CCardBody, CCardHeader,

CTable, CTableHead, CTableRow, CTableHeaderCell,

CTableBody, CTableDataCell,

CButton, CForm, CFormInput, CSpinner

} from "@coreui/react";

import { toast } from "react-toastify";

import axios from "axios";

import jsPDF from "jspdf";

*const* API\_URL = "http://localhost:8080/api/products/";

*const* Products = () => {

*const* [state, setState] = useState({

items: [], form: { id: "", reference: "", name: "", price: "" },

loading: true, search: "", page: 1

});

*const* perPage = 5;

*const* fetchItems = *async* () => {

setState(s => ({ **...**s, loading: true }));

try {

*const* res = *await* axios.get(API\_URL);

setState(s => ({ **...**s, items: Array.isArray(res.data)?res.data:res.data.results||[] }));

} catch { toast.error("Error cargando productos"); }

setState(s => ({ **...**s, loading: false }));

};

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

*const* saveItem = *async* e => {

e.preventDefault();

try {

state.form.id

? *await* axios.put(`${API\_URL}${state.form.id}/`, state.form)

: *await* axios.post(API\_URL, state.form);

toast.success(state.form.id?"Producto actualizado":"Producto creado");

setState(s => ({ **...**s, form: {id:"",reference:"",name:"",price:""} }));

fetchItems();

} catch { toast.error("Error guardando producto"); }

};

*const* deleteItem = *async* id => {

if(!window.confirm("¿Eliminar producto?")) *return*;

try { *await* axios.delete(`${API\_URL}${id}/`); toast.success("Producto eliminado"); fetchItems(); }

catch { toast.error("Error eliminando producto"); }

};

*const* exportPDF = () => {

*const* doc = new jsPDF();

doc.text("Reporte de Productos",20,20);

state.items.forEach((p,i)=>doc.text(`${p.reference} - ${p.name} - $${p.price}`,20,30+i\*10));

doc.save("productos.pdf");

};

*const* filtered = state.items.filter(p=>p.reference.toLowerCase().includes(state.search.toLowerCase())||p.name.toLowerCase().includes(state.search.toLowerCase()));

*const* paginated = filtered.slice((state.page-1)\*perPage,state.page\*perPage);

*const* totalPages = Math.ceil(filtered.length/perPage);

if(state.loading) *return* <CSpinner color="primary" />;

*return* (

<CCard>

<CCardHeader>Gestión de Productos</CCardHeader>

<CCardBody>

<CForm onSubmit={saveItem} className="mb-3 d-flex gap-2 flex-wrap">

{["reference","name","price"].map(f=>(

<CFormInput key={f} type={f==="price"?"number":"text"} placeholder={f} value={state.form[f]} onChange={e=>setState(s=>({**...**s,form:{**...**s.form,[f]:e.target.value}}))} required/>

))}

<CButton type="submit" color="success">{state.form.id?"Actualizar":"Crear"}</CButton>

</CForm>

<div className="d-flex justify-content-between mb-2">

<CFormInput placeholder="Buscar..." style={{maxWidth:"300px"}} value={state.search} onChange={e=>setState(s=>({**...**s,search:e.target.value,page:1}))}/>

<CButton color="info" onClick={exportPDF}>Exportar PDF</CButton>

</div>

<CTable striped hover>

<CTableHead>

<CTableRow><CTableHeaderCell>Referencia</CTableHeaderCell><CTableHeaderCell>Nombre</CTableHeaderCell><CTableHeaderCell>Precio</CTableHeaderCell><CTableHeaderCell>Acciones</CTableHeaderCell></CTableRow>

</CTableHead>

<CTableBody>

{paginated.map(p=>(

<CTableRow key={p.id}>

<CTableDataCell>{p.reference}</CTableDataCell>

<CTableDataCell>{p.name}</CTableDataCell>

<CTableDataCell>{p.price}</CTableDataCell>

<CTableDataCell>

<CButton size="sm" color="warning" className="me-2" onClick={()=>setState(s=>({**...**s,form:p}))}>Editar</CButton>

<CButton size="sm" color="danger" onClick={()=>deleteItem(p.id)}>Eliminar</CButton>

</CTableDataCell>

</CTableRow>

))}

</CTableBody>

</CTable>

<div className="mt-2 d-flex justify-content-center gap-2">

<CButton color="secondary" disabled={state.page<=1} onClick={()=>setState(s=>({**...**s,page:s.page-1}))}>Anterior</CButton>

<span>Página {state.page} / {totalPages||1}</span>

<CButton color="secondary" disabled={state.page>=totalPages} onClick={()=>setState(s=>({**...**s,page:s.page+1}))}>Siguiente</CButton>

</div>

</CCardBody>

</CCard>

);

};

export default Products;

### **Registrar la ruta**

En el archivo src/routes.js agrega tu vista al arreglo de rutas:

import Products from './views/products/Products'

const routes = [

{ path: '/', exact: true, name: 'Home' },

{ path: '/dashboard', name: 'Dashboard', element: Dashboard },

{ path: '/products', name: 'Productos', element: Products}, // 👈 aquí tu componente

]

export default routes

### **4️⃣ Agregar al menú lateral**

El menú lateral de CoreUI se define en src/\_nav.js.  
 Ahí agrega un ítem para que aparezca en el sidebar:

const \_nav = [

{

component: CNavItem,

name: 'Dashboard',

to: '/dashboard',

icon: <CIcon icon={cilSpeedometer} customClassName="nav-icon" />,

},

{

component: CNavItem,

name: 'Productos',

to: '/products',

icon: <CIcon icon={cilCarAlt} customClassName="nav-icon" />, // 👈 ícono opcional

},

]

export default \_nav

⚠️ Si quieres usar cilCarAlt necesitas importar los íconos:

import CIcon from '@coreui/icons-react'

import { cilCarAlt } from '@coreui/icons'

Products.js usa **axios, toastify y jspdf**, por eso se instalaron en la plantilla:

Dentro de [App.js](http://app.js) agregar:

import { ToastContainer } from 'react-toastify'

import 'react-toastify/dist/ReactToastify.css'

{*/\* 🔔 Notificaciones globales \*/*}

<ToastContainer position="top-right" autoClose={3000} />

## **6. Levantar todo**

Desde la raíz (project/):

docker-compose up --build

* Backend →<http://localhost:8000/api/products/>
* Frontend → [http://localhost:3030/#/products](http://localhost:3000/#/products)

npm install jspdf jspdf-autotable