1) Preparación del entorno

Windows 10/11

1. **Node 18+**: https://nodejs.org
2. **Angular & Ionic CLI**

   ```
   npm i -g @angular/cli @ionic/cli
   ```

3. **Android Studio** (SDK + emulador).
4. **Python 3.10+**: https://www.python.org/downloads/
5. **Dependencias C++ p/dlib** (necesarias para `face_recognition`):
   - **CMake**: https://cmake.org/download/
   - **Visual Studio 2022 Build Tools** (c++): https://visualstudio.microsoft.com/downloads/
6. **Docker Desktop**: https://www.docker.com/products/docker-desktop
7. **Opcional (atajo Windows)**: en lugar de compilar dlib, suele funcionar:

   ```
   pip install dlib-bin
   ```

   y luego `pip install face_recognition`. Si dlib-bin no está disponible para tu versión, usa el camino con Build Tools.

Ubuntu 22.04

```
sudo apt update
sudo apt install -y build-essential cmake libopenblas-dev liblapack-dev libjpeg-dev python3-venv
# Node 18 (via nvm recomendado) + Android Studio + Docker Engine
```

---

2) Backend FastAPI + face_recognition + SQLite

Estructura rápida

```
backend/
  ├─ main.py
```

```
    ├─ requirements.txt
    ├─ data/            # se crea sola (SQLite aquí)
    ├─ Dockerfile
    └─ docker-compose.yml
```

requirements.txt

```
fastapi
uvicorn[standard]
face_recognition
numpy
SQLAlchemy>=2
pillow
python-multipart
```

> Si en Windows usas el atajo `dlib-bin`, agrégalo a `requirements.txt` antes de `face_recognition`.

main.py (único archivo)

```
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from typing import Optional, List, Literal
import base64, io, os, datetime
from PIL import Image
import numpy as np
import face_recognition

from sqlalchemy import create_engine, Column, Integer, String, Float, DateTime,
ForeignKey, select, text
from sqlalchemy.orm import declarative_base, Session, relationship

# --- Config ---
DB_DIR = os.path.join(os.path.dirname(__file__), "data")
os.makedirs(DB_DIR, exist_ok=True)
DB_PATH = os.path.join(DB_DIR, "db.sqlite3")
engine = create_engine(f"sqlite:///{DB_PATH}", future=True)
Base = declarative_base()

# --- Modelos DB ---
class Employee(Base):
    __tablename__ = "employees"
    id = Column(String, primary_key=True)  # employee_id definido por el cliente
    name = Column(String, nullable=False)
    # encoding almacenado como texto JSON de floats separados por coma (simple
para POC)
    encoding = Column(String, nullable=False)  # "0.12,0.34,..."

    logs = relationship("AccessLog", back_populates="employee")
```

```
class AccessLog(Base):
    __tablename__ = "access_logs"
    id = Column(Integer, primary_key=True, autoincrement=True)
    employee_id = Column(String, ForeignKey("employees.id"), nullable=False)
    event = Column(String, nullable=False)  # "in" | "out"
    ts = Column(DateTime, default=datetime.datetime.utcnow)

    employee = relationship("Employee", back_populates="logs")

Base.metadata.create_all(engine)

# --- FastAPI ---
app = FastAPI(title="Employee Face POC")

origins = [
    "http://localhost",
    "http://localhost:8100",      # Ionic serve
    "capacitor://localhost",
    "ionic://localhost",
    "http://10.0.2.2:8100",       # Emulador Android (opcional)
]
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# --- Schemas ---
class RegisterFaceReq(BaseModel):
    employee_id: str
    name: str
    image_base64: str  # sin el prefijo data:image/...

class RegisterFaceRes(BaseModel):
    status: Literal["ok"]
    employee_id: str

class CheckReq(BaseModel):
    image_base64: str

class CheckRes(BaseModel):
    recognized: bool
    employee_id: Optional[str] = None
    name: Optional[str] = None
    distance: Optional[float] = None
    event: Optional[Literal["in","out"]] = None
    ts: Optional[str] = None

# --- Utilidades ---
def b64_to_rgb_np(b64: str) -> np.ndarray:
    img_bytes = base64.b64decode(b64)
    img = Image.open(io.BytesIO(img_bytes)).convert("RGB")
    return np.array(img)

def compute_encoding(b64: str) -> List[float]:
    image_np = b64_to_rgb_np(b64)
    boxes = face_recognition.face_locations(image_np, model="hog")  # HOG para POC
(sin GPU)
    if not boxes:
        raise HTTPException(status_code=422, detail="No se detectó rostro en la
imagen.")
    encs = face_recognition.face_encodings(image_np, boxes)
    if not encs:
        raise HTTPException(status_code=422, detail="No se pudo extraer el
encoding del rostro.")
    return encs[0].tolist()

def serialize_encoding(enc: List[float]) -> str:
    return ",".join(f"{v:.8f}" for v in enc)

def deserialize_encoding(s: str) -> np.ndarray:
    return np.array([float(x) for x in s.split(",")], dtype=np.float32)

TOLERANCE = 0.6

def decide_event(session: Session, employee_id: str) -> str:
    last = session.execute(
        select(AccessLog).where(AccessLog.employee_id ==
employee_id).order_by(AccessLog.ts.desc()).limit(1)
    ).scalar_one_or_none()
    return "out" if (last and last.event == "in") else "in"

# --- Endpoints ---
@app.get("/health")
def health():
    return {"status": "ok"}

@app.post("/register_face", response_model=RegisterFaceRes)
def register_face(req: RegisterFaceReq):
    enc = compute_encoding(req.image_base64)
    enc_s = serialize_encoding(enc)
    with Session(engine) as session:
        emp = session.get(Employee, req.employee_id)
        if emp:
            emp.name = req.name
            emp.encoding = enc_s
        else:
            emp = Employee(id=req.employee_id, name=req.name, encoding=enc_s)
            session.add(emp)
        session.commit()
    return {"status": "ok", "employee_id": req.employee_id}

@app.post("/check_in_out", response_model=CheckRes)
def check_in_out(req: CheckReq):
    probe = np.array(compute_encoding(req.image_base64), dtype=np.float32)
    with Session(engine) as session:
        employees = session.execute(select(Employee)).scalars().all()
        if not employees:
            raise HTTPException(status_code=400, detail="No hay empleados
registrados.")

        best_id, best_name, best_dist = None, None, 1e9
        for e in employees:
            db_enc = deserialize_encoding(e.encoding)
            # Distancia euclidiana (face_recognition usa distancia similar)
            dist = np.linalg.norm(db_enc - probe)
            if dist < best_dist:
                best_id, best_name, best_dist = e.id, e.name, dist

        if best_dist > TOLERANCE:
            return {"recognized": False}

        event = decide_event(session, best_id)
        log = AccessLog(employee_id=best_id, event=event)
        session.add(log)
        session.commit()

        return {
            "recognized": True,
            "employee_id": best_id,
            "name": best_name,
            "distance": float(best_dist),
            "event": event,
            "ts": log.ts.isoformat()
        }

@app.get("/employees")
def list_employees():
    with Session(engine) as session:
        rows = session.execute(select(Employee)).scalars().all()
        return [{"employee_id": r.id, "name": r.name} for r in rows]

@app.get("/logs")
def list_logs():
    with Session(engine) as session:
        rows = session.execute(
            select(AccessLog).order_by(AccessLog.ts.desc()).limit(100)
        ).scalars().all()
        return [{"id": r.id, "employee_id": r.employee_id, "event": r.event, "ts":
r.ts.isoformat()} for r in rows]
```

Probar local

```
cd backend
python -m venv venv
```

```
# Windows:
venv\Scripts\activate
# Linux/macOS:
source venv/bin/activate

pip install --upgrade pip
pip install -r requirements.txt
# (Windows atajo) si falla dlib:
# pip install dlib-bin && pip install face_recognition

uvicorn main:app --reload --host 0.0.0.0 --port 8000
# http://localhost:8000/docs
```

---

3) Dockerizar el backend

Dockerfile

```
FROM python:3.11-bullseye

# Dependencias de compilación para dlib/face_recognition
RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential cmake libopenblas-dev liblapack-dev libjpeg-dev \
 && rm -rf /var/lib/apt/lists/*

WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir --upgrade pip \
 && pip install --no-cache-dir -r requirements.txt

COPY . .
# Crear carpeta de datos para SQLite
RUN mkdir -p /app/data

EXPOSE 8000
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

docker-compose.yml

```
services:
  api:
    build: .
    image: employee-backend:latest
    container_name: employee-backend
    ports:
      - "8000:8000"
    volumes:
      - ./data:/app/data   # Persistir SQLite
    restart: unless-stopped
```

Build & run

```
cd backend
docker compose up --build
# http://localhost:8000/docs
```

---

4) Frontend Ionic/Angular + Capacitor (Android)

Crear proyecto base

```
ionic start employee-register blank --type=angular
cd employee-register
ionic integrations enable capacitor
npm i @capacitor/camera
npm i @capacitor/android
npx cap add android
```

Permisos de cámara (Android)

En `android/app/src/main/AndroidManifest.xml`, agrega:

```
<uses-permission android:name="android.permission.CAMERA" />
```

(Red local/Emulador) Cleartext & Network Security

Si usarás HTTP (no HTTPS) hacia el backend:

- En `AndroidManifest.xml`, dentro de `<application ...>` agrega:

```
android:usesCleartextTraffic="true"
android:networkSecurityConfig="@xml/network_security_config"
```

- Crea `android/app/src/main/res/xml/network_security_config.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <domain-config cleartextTrafficPermitted="true">
```

```
    <domain includeSubdomains="true">10.0.2.2</domain> <!-- Emulador -->
    <domain includeSubdomains="true">192.168.0.0/16</domain> <!-- LAN -->
    <domain includeSubdomains="true">localhost</domain>
  </domain-config>
</network-security-config>
```

> Emulador Android accede al host como **`http://10.0.2.2:8000`**. En dispositivo físico, usa la IP LAN de tu PC (por ejemplo `http://192.168.1.50:8000`).

environments

`src/environments/environment.ts` (dev con Ionic serve):

```
export const environment = {
  production: false,
  apiBaseUrl: 'http://localhost:8000'
};
```

Para emulador:

```
// src/environments/environment.emu.ts
export const environment = {
  production: false,
  apiBaseUrl: 'http://10.0.2.2:8000'
};
```

Servicio API

`src/app/services/api.service.ts`

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { environment } from '../../environments/environment';

@Injectable({ providedIn: 'root' })
export class ApiService {
  private base = environment.apiBaseUrl;

  constructor(private http: HttpClient) {}

  registerFace(employee_id: string, name: string, image_base64: string) {
    return this.http.post(`${this.base}/register_face`, { employee_id, name,
image_base64 });
  }

  checkInOut(image_base64: string) {
    return this.http.post(`${this.base}/check_in_out`, { image_base64 });
  }

  getLogs() {
    return this.http.get(`${this.base}/logs`);
  }
}
```

Captura con Camera y envío

`src/app/home/home.page.ts`

```
import { Component } from '@angular/core';
import { Camera, CameraResultType, CameraSource } from '@capacitor/camera';
import { ApiService } from '../services/api.service';

@Component({
  selector: 'app-home',
  templateUrl: 'home.page.html',
})
export class HomePage {
  loading = false;
  message = '';

  employee_id = '';
  name = '';

  constructor(private api: ApiService) {}

  async captureBase64(): Promise<string | null> {
    const photo = await Camera.getPhoto({
      quality: 80,
      resultType: CameraResultType.Base64,
      source: CameraSource.Camera
    });
    return photo.base64String ?? null;
  }

  async enroll() {
    this.message = '';
    if (!this.employee_id || !this.name) { this.message = 'Employee ID y Name son
requeridos'; return; }
    this.loading = true;
    try {
      const b64 = await this.captureBase64();
      if (!b64) { this.message = 'No se pudo capturar la imagen'; return; }
      await this.api.registerFace(this.employee_id, this.name, b64).toPromise();
      this.message = `Registrado: ${this.employee_id}`;
    } catch (e: any) {
      this.message = e?.error?.detail || 'Error registrando rostro';
    } finally {
      this.loading = false;
    }
  }

  async check() {
    this.message = '';
    this.loading = true;
    try {
      const b64 = await this.captureBase64();
      if (!b64) { this.message = 'No se pudo capturar la imagen'; return; }
      const res: any = await this.api.checkInOut(b64).toPromise();
      if (res.recognized) {
        this.message = `${res.event?.toUpperCase()} de ${res.name}
(${res.employee_id}) @ ${res.ts}`;
      } else {
        this.message = 'Rostro no reconocido';
      }
    } catch (e: any) {
      this.message = e?.error?.detail || 'Error en verificación';
    } finally {
      this.loading = false;
    }
  }
}
```

`src/app/home/home.page.html`

```
<ion-header>
  <ion-toolbar>
    <ion-title>Registro de Ingreso (POC)</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content class="ion-padding">
  <ion-card>
    <ion-card-header>
      <ion-card-title>Enroll (Registrar Rostro)</ion-card-title>
    </ion-card-header>
    <ion-card-content>
      <ion-item>
        <ion-input label="Employee ID" [(ngModel)]="employee_id"></ion-input>
      </ion-item>
      <ion-item>
        <ion-input label="Name" [(ngModel)]="name"></ion-input>
      </ion-item>
      <ion-button expand="block" (click)="enroll()"
[disabled]="loading">Registrar</ion-button>
    </ion-card-content>
  </ion-card>

  <ion-card>
    <ion-card-header>
      <ion-card-title>Check In/Out</ion-card-title>
    </ion-card-header>
    <ion-card-content>
      <ion-button expand="block" (click)="check()" [disabled]="loading">Tomar Foto
y Verificar</ion-button>
    </ion-card-content>
  </ion-card>

  <ion-text color="medium">
    <p *ngIf="loading">Procesando...</p>
    <p>{{ message }}</p>
  </ion-text>
</ion-content>
```

> Asegúrate de importar `FormsModule` en el módulo de la página para `ngModel`, y `HttpClientModule` en `AppModule`.

Ejecutar

- Web (solo pruebas de UI): `ionic serve` (no tiene cámara real).
- Android Emulador:

  ```
  ionic build
  npx cap sync android
  npx cap run android
  ```

  > Si usas emulador, compila con el `environment.emu.ts` o ajusta `apiBaseUrl` a `http://10.0.2.2:8000`.

- Dispositivo físico:
  - Conecta vía USB, activa "Depuración USB".
  - Asegúrate de que la IP del backend sea la IP LAN de tu PC.

---

5) Integración y pruebas

Postman / cURL (API local)

- Registro (reemplaza `<B64>` con base64 de una foto):

```
curl -X POST http://localhost:8000/register_face \
  -H "Content-Type: application/json" \
  -d '{"employee_id":"E001","name":"Alice","image_base64":"<B64>"}'
```

- Check In/Out:

```
curl -X POST http://localhost:8000/check_in_out \
  -H "Content-Type: application/json" \
  -d '{"image_base64":"<B64>"}'
```

- Ver logs:

```
curl http://localhost:8000/logs
```

Validar en SQLite

El archivo queda en `backend/data/db.sqlite3`. Puedes abrirlo con DB Browser for SQLite o:

```
sqlite3 backend/data/db.sqlite3 "SELECT * FROM access_logs ORDER BY ts DESC LIMIT 5;"
```

---

6) Gotchas (POC)

- **Calidad de imagen**: usa buena iluminación y encuadre frontal.
- **Un rostro por foto** (el backend toma el primero).
- **CORS**: ya configurado para `localhost`, `ionic://localhost` y `capacitor://localhost`.
- **Distancia/Tolerancia**: ajusta `TOLERANCE = 0.6` según resultados (0.5 más estricto, 0.65 más laxo).
- **Rendimiento**: `model="hog"` es CPU-friendly. Para producción, evalúa CNN con GPU (no en este POC).
- **Docker**: la imagen compila dlib; tarda un poco la primera vez.

---

7) Checklist de entregables (según tu JSON)

- ☑ **App Ionic** con captura de foto y envío (base64) al backend.
- ☑ **Backend FastAPI** con:
  - `POST /register_face`
  - `POST /check_in_out`
  - `GET /employees`, `GET /logs` (útiles para pruebas)
- ☑ **SQLite** en `backend/data/db.sqlite3` con registros de **ingresos/egresos**.
- ☑ **Dockerfile** y **docker-compose.yml** para desplegar el backend en contenedor.

Si quieres, luego te dejo un script de **datos de prueba** y un pequeño **seed** de empleados, o agregamos un endpoint `DELETE /employees/{id}` para re-entrenar/limpiar.