StatuApp

Repositorio del proyecto **StatuApp**: catálogo digital de estatuas y monumentos de Rosario con acceso por **QR** y ficha ampliada.

> Objetivo inicial: levantar el **backend en Django/DRF** y el **frontend en Next.js** de forma local, sin Docker, con 1–2 fichas de ejemplo y navegación por slug.

---

1) Arquitectura (visión rápida)

- **Frontend**: Next.js 14 (App Router), TypeScript, ISR/SSG para fichas, mapa con React Leaflet.
- **Backend**: Django + Django REST Framework.
- **Base de datos**: MySQL/MariaDB (en esta primera etapa podés usar SQLite si preferís simplicidad).
- **Contenido**: cada estatua se identifica por un **slug** (ej: `mujer-con-nino-rosedal`).

```
frontend  →  consume API REST  →  backend (Django/DRF)  →  BD
```

---

2) Requisitos

- **Node.js** LTS (>= 18)
- **npm** (o pnpm/yarn)
- **Python** 3.10+
- **MySQL/MariaDB** (opcional si preferís comenzar con SQLite)

> Windows: recomendamos PowerShell. Si `python` no está en PATH, usá `py -3`.

---

3) Estructura del repo (sugerida)

```
statuApp/
├─ backend/            # Django + DRF
├─ frontend/           # Next.js + TS
├─ db/                 # esquema SQL, DER, notas
├─ presentacion/       # PDFs y material del proyecto
└─ README.md           # este archivo
```

---

4) Puesta en marcha — Backend (Django/DRF)

1. **Entrar a la carpeta**:

```
cd backend
```

2. **Crear y activar venv**:
3. Windows (PowerShell):

```
py -3 -m venv .venv
.\.venv\Scripts\Activate
```

4. Linux/Mac:

```
python3 -m venv .venv
source .venv/bin/activate
```

5. **Instalar dependencias**:

```
pip install -r requirements.txt
```

6. **Configurar variables de entorno** (crear `.env` en `backend/`):

```
# Si usás SQLite para empezar (simple):
DJANGO_SECRET_KEY=changeme
DEBUG=True
DATABASE_URL=sqlite:///db.sqlite3

# Si usás MySQL (opcional):
# DATABASE_URL=mysql://usuario:password@localhost:3306/statuapp
# CORS_ALLOWED_ORIGINS=http://localhost:3000
```

7. **Migraciones y superusuario**:

```
python manage.py migrate
python manage.py createsuperuser
```

8. **(Opcional) Cargar datos de ejemplo**: ver sección **Datos de ejemplo**.
9. **Levantar el server**:

```
python manage.py runserver
```

La API quedará en: `http://127.0.0.1:8000/`

Endpoints mínimos esperados

- `GET /api/v1/statues/` → lista con filtros (`?q=`, `?barrio=`, `?tag=`)
- `GET /api/v1/statues/{slug}/` → ficha completa por slug
- `POST /api/v1/events/scan` → registrar lectura de QR (más adelante)

Solución de errores comunes - *"no such table ..."*: corré `python manage.py makemigrations && python manage.py migrate`. - *CORS al llamar desde el front*: agregá `CORS_ALLOWED_ORIGINS=http://localhost:3000` y habilitá `django-cors-headers`.

---

5) Puesta en marcha — Frontend (Next.js)

1. **Entrar a la carpeta**:

```
cd frontend
```

2. **Instalar dependencias**:

```
npm install
```

3. **Variables de entorno**: crear `frontend/.env.local` con:

```
NEXT_PUBLIC_API_BASE=http://127.0.0.1:8000/api/v1
NEXT_PUBLIC_SITE_URL=http://localhost:3000
```

4. **Levantar en desarrollo**:

```
npm run dev
```

Abrí `http://localhost:3000`.

Nota: si te aparece "`next` no se reconoce" en Windows, asegurate de correr **los scripts de npm** (`npm run dev`) en lugar de ejecutar `next` directo. `next` se instala local al proyecto.

Rutas del front (mínimas)

- `/` → Home/landing
- `/s/[slug]` → Ficha de estatua (carga con `fetch` a la API)
- `/mapa` → Mapa con marcadores (luego)
- `/buscar` → Buscador y filtros (luego)

Componente de datos

- `frontend/lib/api.ts` con un `fetchJson` que use `NEXT_PUBLIC_API_BASE`.
- Páginas con **ISR** (`export const revalidate = 60`) para fichas.

---

6) Datos de ejemplo (seed)

Opción A: Django fixtures

1. Crear `backend/fixtures/statues.json` con 1–2 estatuas:

```
[
  {
    "model": "statues.statue",
    "pk": 1,
    "fields": {
      "slug": "mujer-con-nino-rosedal",
      "title": "Mujer con Niño (Rosedal)",
      "lat": -32.9566,
      "lng": -60.6604,
      "barrio": "Parque Independencia",
      "description_md": "Escultura ubicada en el Rosedal del Parque
Independencia.",
      "published": true
    }
  }
]
```

2. Cargar fixture:

```
python manage.py loaddata fixtures/statues.json
```

Opción B: SQL (si usás MySQL)

- Importá `db/statuapp_mysql_schema.sql`.
- Insertá 1–2 filas en `estatuas` (o tabla equivalente) y asegurá **slug** único.

---

7) QR: convención de URL

- Formato: `https://tu-dominio/s/{slug}`
- En desarrollo: `http://localhost:3000/s/{slug}`
- El QR apunta a esa URL y el front resuelve el contenido llamando a la API.

---

8) Scripts útiles

Backend

```
# desde backend/
pip freeze > requirements.txt
python manage.py makemigrations
```

```
python manage.py migrate
python manage.py runserver
```

Frontend

```
# desde frontend/
npm run dev
npm run build
npm run start
npm run lint
```

---

9) Convenciones y calidad

- **Backend**: `black`, `isort`, `ruff` (opcional).
- **Frontend**: `eslint` + `prettier`.
- **Commits**: mensajes claros; ejemplo `feat(front): ficha /s/[slug]`.

---

10) Roadmap corto (MVP)

1. Render de `/s/[slug]` con datos reales (1–2 estatuas).
2. Home con CTA + link al mapa.
3. Listado `/buscar` con barra de búsqueda (client-side) y filtros simples.
4. Mapa `/mapa` con marcador(es) inicial(es).
5. Métricas de QR (`POST /events/scan`) más adelante.

---

11) Licencia y créditos

- Licencia: por definir (MIT recomendada si querés apertura).
- Créditos: autoría de textos/fotos, fuentes consultadas y relevamientos.

---

12) Problemas frecuentes (FAQ)

- `next` **no se reconoce**: usá `npm run dev`. Si falla, `npm install` primero.
- `no such table` **en Django**: corré `makemigrations` y `migrate`.
- **CORS bloquea el fetch**: agregá el origen del front a `CORS_ALLOWED_ORIGINS`.
- **Variables de entorno**: verificá `.env` en `backend/` y `.env.local` en `frontend/`.

¡Listo! Con esto podés levantar todo de manera local y sencilla, sin Docker. Cuando estés lista, armamos la configuración para deploy y contenedores.