Statu App — Arquitectura Técnica & Stack

Propuesta técnica para la implementación web con QR
Un proyecto de Ionline.com.ar

Este documento describe el enfoque de ingeniería para **Statu App**, una plataforma web que permite escanear códigos QR colocados en estatuas de Rosario para acceder a fichas informativas ricas en texto, imágenes y mapa. Se presentan el stack recomendado, el modelo de datos inicial, endpoints, despliegue, seguridad y un roadmap técnico.

1) Arquitectura de Alto Nivel

La solución seguirá una arquitectura **web clásica client–server** con separación de responsabilidades: • **Frontend**: React + Next.js (SSR/SSG para SEO y performance).
• **Backend API**: Django + Django REST Framework (DRF) para exponer endpoints JSON y panel admin.
• **Base de Datos**: PostgreSQL (recomendado en prod) o MySQL (si ya está disponible).
• **Storage estático**: S3 compatible (o CDN) para imágenes de estatuas y QR.
• **Autenticación**: JWT o Session Auth (según si habrá app móvil futura).
• **Infra**: Docker + docker-compose en dev; en prod: VPS / PaaS (Railway/Render/Heroku-like) o Kubernetes si escala.
• **Observabilidad**: logs centralizados + métricas y alertas.

```
[ Usuario / Móvil ] ■ (HTTPS, QR -> URL) ▼ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
Next.js (Frontend) ■ ← CDN ■ - Páginas públicas ■ ■ - SSR/SSG + SEO ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ (HTTPS, JSON) ▼ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Django + DRF (API) ■ ■ - Admin / CRUD ■ ■ - Autenticación ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ ■ (SQL) ■ ■ (Objetos) ▼ ▼ [PostgreSQL] [S3/MinIO]
```

2) Stack Tecnológico Sugerido

Capa	Tecnología	Motivo
Frontend	React 18 + Next.js 14 (App Router)	SSR/SSG, SEO, rutas fáciles, performance.
Estilos UI	TailwindCSS + Headless UI	Velocidad, consistencia y accesibilidad.
Mapas	Leaflet + react-leaflet / Mapbox	Mapa interactivo de estatuas.
Backend	Python 3.12 + Django 5 + DRF	Rapidez de desarrollo; admin potente.
BD	PostgreSQL 15 (prod) / SQLite (dev)	Robustez, geodatos, JSON fields.
Storage	AWS S3 / MinIO	Imágenes/documentos; CDN.
Auth	djangorestframework-simplejwt	Estándar, escalable a móvil.
Cache	Redis	Rate limit, cache de respuestas, colas.
DevOps	Docker, docker-compose	Entornos reproducibles.
CI/CD	GitHub Actions	Tests, build y deploy automatizados.

3) Modelo de Datos (MVP)

Entidad principal: **Statue** (Estatua). Otras entidades: **Author**, **Location**, **Media**, **Tag**.
Campos clave (sugerencia): • Statue: id (UUID), slug, title, description_md, year, material, barrio, lat, lng, author_id, created_at, updated_at, is_published.
• Author: id, name, birth_year, death_year, bio_md.
• Location: id, name, address, barrio, lat, lng, notes_md.
• Media: id, statue_id, kind (photo|audio|doc), url, caption, credit.
• Tag: id, name; relación many■to■many statue_tags.

```
Statue(id, slug, title, description_md, year, material, barrio, lat, lng, author_id, location_id, is_published, created_at, updated_at) Author(id, name, birth_year, death_year, bio_md) Location(id, name, address, barrio, lat, lng, notes_md) Media(id, statue_id, kind, url, caption, credit) Tag(id, name) StatueTag(statue_id, tag_id)
```

4) API REST (DRF) — Endpoints Iniciales

GET /api/v1/statues/ — lista paginada con filtros (q, barrio, tag, author, published).
GET /api/v1/statues/{slug}/ — detalle con author, location y media.
POST /api/v1/statues/ — crear (autenticado, rol editor).
PUT/PATCH /api/v1/statues/{slug}/ — actualizar (editor).
DELETE /api/v1/statues/{slug}/ — eliminar (admin).
GET /api/v1/tags/ — lista de tags.
GET /api/v1/authors/ — lista + detalle.
Auth: /api/token/ y /api/token/refresh/ (JWT).

5) Flujo del QR

1) Se genera un QR con la URL canónica: **https://statu.app/s/{slug}**.
2) El usuario escanea y llega a una **página Next.js** (SSG/ISR) que prerenderiza la ficha.
3) La página consulta la API si necesita datos frescos (ISR) y muestra: título, autor, barrio, mapa y galería.
4) Métricas de escaneo: UTM/source + endpoint de tracking (/api/v1/events/scan).

6) SEO, Performance y Accesibilidad

• **Next.js SSG/ISR** para fichas: HTML indexable, meta tags por estatua (OpenGraph/Twitter).
• **Mapa** con lazy loading, imágenes optimizadas (next/image).
• **i18n** futuro (es/en).
• **WCAG AA**: contraste, etiquetas aria, teclado, transcripciones de audio.

7) Seguridad

• HTTPS forzado, HSTS, CSRF en panel admin, JWT con expiración corta y refresh.
• Roles: admin, editor (carga de contenido), lector (público).
• Validación de subida de archivos (extensiones, tamaño, antivirus opcional).
• Rate limit por IP (Nginx/DRF throttling) para /api y /auth.
• Backups automáticos de BD + versionado de media.

8) Despliegue

• **Dev**: docker-compose (web, api, db, redis, minio).
• **Prod**: VPS (Ubuntu 22.04), Nginx como reverse proxy, Gunicorn para Django, PM2/Node para Next.js (o Vercel).
• **CI/CD**: GitHub Actions con jobs de test, build y deploy. Migraciones automáticas y collectstatic.

9) Variables de Entorno (ejemplo)

DJANGO_SECRET_KEY, DJANGO_DEBUG, DATABASE_URL, ALLOWED_HOSTS, AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, S3_BUCKET, S3_REGION, REDIS_URL, JWT_SECRET, NEXT_PUBLIC_API_BASE, NEXT_PUBLIC_MAP_TOKEN.

10) Roadmap Técnico (MVP → v1.0)

Fase	Alcance	Duración Est.
MVP (v0.1)	CRUD de estatuas, fichas públicas SSG, QR funcional, admin.	2–4 semanas
v0.2	Mapa, filtros por barrio/tags, galería de imágenes.	2 semanas
v0.3	Analytics de escaneos, performance, SEO avanzado.	1–2 semanas
v1.0	AR/Audio guías opcionales, i18n, onboarding de escuelas.	3–6 semanas

11) Quickstart de Desarrollo (resumen)

Backend
• Crear venv, instalar Django 5 y DRF. `django-admin startproject statu_api`.
• App `statues` con modelos anteriores; correr migraciones; crear superuser; DRF routers.
• Cargar datos seed (fixtures) y activar CORS.

Frontend
• `npx create-next-app@latest statu-web` (App Router).
• Páginas: `/s/[slug]`, `/mapa`, `/buscar`; fetch a **NEXT_PUBLIC_API_BASE**.
• Imagenes optimizadas, metadatos por estatua, mapa con react-leaflet.

Anexo A — Modelo Django (ejemplo abreviado)

```
class Statue(models.Model): id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False) slug = models.SlugField(unique=True) title = models.CharField(max_length=180) description_md = models.TextField(blank=True) year = models.IntegerField(null=True, blank=True) material = models.CharField(max_length=120, blank=True) barrio = models.CharField(max_length=120, blank=True) lat = models.DecimalField(max_digits=9, decimal_places=6, null=True, blank=True) lng = models.DecimalField(max_digits=9, decimal_places=6, null=True, blank=True) author = models.ForeignKey(Author, on_delete=models.SET_NULL, null=True, blank=True) location = models.ForeignKey(Location, on_delete=models.SET_NULL, null=True, blank=True) is_published = models.BooleanField(default=True) created_at = models.DateTimeField(auto_now_add=True) updated_at = models.DateTimeField(auto_now=True)
```

Anexo B — Next.js Ruta de Ficha (concepto)

```
// app/s/[slug]/page.tsx export default async function StatuePage({ params }) { const res = await fetch(`${process.env.NEXT_PUBLIC_API_BASE}/api/v1/statues/${params.slug}`, { next: { revalidate: 60 }}); const data = await res.json(); return ( {data.title} {data.barrio} • {data.year ?? 's/f'} {/* Mapa e imágenes */} ); }
```