**Architecture Decision Records (ADR)**

**Plataforma Arty**

**Proyecto:** Plataforma Arty

**Equipo:** Angel Quishpe, Jorge Escobar, Alan Rivera

**Fecha de inicio:** 20 de octubre de 2025

Índice

[ADR-001: Inicio con Arquitectura Monolítica Modular 2](#_Toc211972239)

[ADR-002: Selección de PostgreSQL como Base de Datos Principal 3](#_Toc211972241)

[ADR-003: Uso de AWS S3 para Almacenamiento de Imágenes 4](#_Toc211972243)

[ADR-004: Implementación de Autenticación con JWT 5](#_Toc211972245)

[ADR-005: Arquitectura de Frontend con React y React Native 6](#_Toc211972246)

[ADR-006: Implementación Inicial de Subastas Dentro del Monolito 7](#_Toc211972247)

[ADR-007: Integración con Servicios Externos para Funcionalidades Críticas 9](#_Toc211972248)

[ADR-008: Estrategia de Despliegue con Contenedores Docker 10](#_Toc211972249)

# ADR-001: Inicio con Arquitectura Monolítica Modular

# 

**Estado:** Aceptado

**Fecha:** 2025-10-21

**Autores:** Angel Quishpe, Alan Rivera, Jorge Escobar

**Contexto**

La Plataforma Arty está en fase inicial (MVP) con un equipo pequeño y necesita lanzar rápidamente al mercado. No tenemos certeza sobre los patrones de uso ni los puntos de mayor demanda. Prematuramente adoptar microservicios agregaría complejidad innecesaria y ralentizaría el desarrollo.

**Decisión**

Comenzaremos con una arquitectura monolítica modular en Spring Boot, diseñada con separación clara de responsabilidades (capas y módulos) que facilite una futura migración incremental a microservicios cuando la demanda lo justifique.

**Criterios de migración a microservicios:**

* Más de 100,000 usuarios activos mensuales
* Cuellos de botella claros de performance identificados
* Equipos de desarrollo que crezcan a 3+ equipos
* Necesidad de escalar componentes específicos independientemente
* Alternativas Consideradas
* **Microservicios desde el inicio:** Complejidad prematura
* **Monolito sin modularizarían:** Dificulta migración futura
* **Serverless puro:** No adecuado para lógica de negocio compleja

**Consecuencias**

**Positivas:**

* Desarrollo más rápido del MVP
* Deployment simple (un solo artefacto)
* Debugging más fácil
* Transacciones ACID simples sin coordinación distribuida
* Menor complejidad operacional
* Costo de infraestructura reducido inicialmente
* Un solo código base facilita cambios rápidos

**Negativas:**

* Escalamiento horizontal más limitado inicialmente
* Todo el sistema debe desplegarse junto
* Riesgo de acoplamiento si no se mantiene disciplina modular
* Un error puede afectar todo el sistema
* Plan de Migración Futura

**Cuando se cumplan los criterios, migraremos incrementalmente:**

* **Fase 1:** Extraer servicio de Subastas (tiempo real crítico)
* **Fase 2:** Extraer servicio de Pagos/Transacciones (seguridad y compliance)
* **Fase 3:** Extraer servicios según análisis de carga real

# ADR-002: Selección de PostgreSQL como Base de Datos Principal

# 

**Estado:** Aceptado

**Fecha:** 2025-10-21

**Autores:** Angel Quishpe, Alan Rivera, Jorge Escobar

**Contexto**

Necesitamos una base de datos que soporte:

* Relaciones complejas entre usuarios, obras, galerías y transacciones
* Transacciones ACID para operaciones de compra/venta
* Consultas complejas con joins
* Escalabilidad vertical y horizontal
* Integridad referencial estricta

**Decisión**

Seleccionamos PostgreSQL como base de datos relacional principal.

**Alternativas Consideradas**

* MySQL: Menos características avanzadas, menor soporte para JSON
* MongoDB: No relacional, dificulta integridad referencial crítica para transacciones
* Oracle: Costos de licenciamiento prohibitivos

**Consecuencias**

**Positivas:**

* Soporte robusto para transacciones ACID
* Excelente rendimiento en consultas complejas
* Soporte nativo para JSON (flexibilidad futura)
* Open source con comunidad activa
* Capacidades de búsqueda full-text
* Extensiones avanzadas (PostGIS para geolocalización futura)

**Negativas:**

* Requiere optimización de índices para escalar
* Complejidad en sharding horizontal comparado con NoSQL

# ADR-003: Uso de AWS S3 para Almacenamiento de Imágenes

# 

**Estado:** Aceptado

**Fecha:** 2025-10-21

**Autores:** Angel Quishpe, Alan Rivera, Jorge Escobar

**Contexto**

Las obras de arte requieren imágenes de alta resolución que pueden ser muy pesadas. Almacenarlas en la base de datos afectaría negativamente el rendimiento y los costos. Necesitamos una solución escalable, confiable y con CDN integrado.

**Decisión**

Utilizaremos Amazon S3 con CloudFront CDN para almacenar y servir imágenes de obras de arte.

**Alternativas Consideradas**

* **Almacenamiento en servidor local:** No escalable, riesgos de pérdida de datos
* **Google Cloud Storage:** Similar a S3, menor ecosistema
* **Cloudinary:** Más costoso, demasiadas features innecesarias

**Consecuencias**

**Positivas:**

* Almacenamiento prácticamente ilimitado
* Alta disponibilidad (99.99%)
* Integración con CDN para entrega rápida global
* Versionado de archivos
* Políticas de acceso granulares
* Costos por uso (pay-as-you-go)
* Procesamiento de imágenes bajo demanda

**Negativas:**

* Costos de transferencia de datos
* Dependencia de proveedor cloud (vendor lock-in)
* Latencia en primera carga sin CDN

# ADR-004: Implementación de Autenticación con JWT

**Estado:** Aceptado

**Fecha:** 2025-10-21

**Autores:** Angel Quishpe, Alan Rivera, Jorge Escobar

**Contexto**

Necesitamos un mecanismo de autenticación que funcione bien con arquitectura REST API, sea escalable, y permita autenticación stateless entre múltiples servicios (API principal y servicio de subastas).

**Decisión**

Implementaremos autenticación basada en JSON Web Tokens (JWT) con:

* Tokens de acceso de corta duración (15 minutos)
* Refresh tokens de larga duración (7 días)
* Almacenamiento seguro en httpOnly cookies

**Alternativas Consideradas**

* **Sesiones en servidor:** No escalable horizontalmente, requiere sticky sessions
* **OAuth 2.0 completo**: Demasiado complejo para MVP
* **API Keys:** Menos seguro, sin expiración

**Consecuencias**

**Positivas:**

* Stateless: no requiere almacenamiento en servidor
* Escalable horizontalmente sin problemas
* Estándar de industria bien documentado
* Permite compartir autenticación entre servicios
* Self-contained: toda la info está en el token

**Negativas:**

* No se puede revocar inmediatamente (hasta expiración)
* Tamaño del token mayor que session ID
* Requiere gestión segura de claves secretas
* Necesita estrategia de rotación de tokens

# ADR-005: Arquitectura de Frontend con React y React Native

**Estado:** Aceptado

**Fecha:** 2025-10-21

**Autores:** Angel Quishpe, Alan Rivera, Jorge Escobar

**Contexto**

Necesitamos presencia en web y móvil (iOS/Android). Buscamos maximizar la reutilización de código, acelerar el desarrollo y mantener una experiencia de usuario consistente.

**Decisión**

**Usaremos:**

* React con TypeScript para la aplicación web
* React Native para aplicaciones móviles iOS y Android
* Componentes compartidos cuando sea posible

**Alternativas Consideradas**

* **Flutter:** Curva de aprendizaje de Dart, menor ecosistema web
* **Apps nativas separadas:** Mayor costo de desarrollo y mantenimiento
* **Progressive Web App (PWA):** Limitaciones en features nativas móviles

**Consecuencias**

**Positivas:**

* Reutilización de lógica de negocio y componentes
* Un solo lenguaje (JavaScript/TypeScript) en frontend
* Gran comunidad y ecosistema de librerías
* Hot reload acelera desarrollo
* Experiencia cercana a nativa en móviles

**Negativas:**

* Performance ligeramente inferior a apps nativas
* Tamaño de bundle más grande
* Algunas limitaciones en features específicas de plataforma
* Actualizaciones de React Native pueden romper compatibilidad

# ADR-006: Implementación Inicial de Subastas Dentro del Monolito

**Estado:** Aceptado

**Fecha:** 2025-10-21

**Autores:** Angel Quishpe, Alan Rivera, Jorge Escobar

**Contexto**

Las subastas requieren actualizaciones en tiempo real. Inicialmente para el MVP, esperamos volumen bajo de subastas concurrentes (< 50 simultáneas). Implementar un servicio separado con WebSockets agregaría complejidad operacional prematura.

**Decisión**

Implementaremos las subastas dentro del monolito Spring Boot usando:

* Server-Sent Events (SSE) o WebSockets con Spring WebSocket
* Actualización de pujas mediante endpoints REST + notificaciones push
* **Migración futura:** Cuando tengamos >100 subastas concurrentes o problemas de latencia, extraeremos a un servicio independiente con Node.js + Socket.io

**Alternativas Consideradas**

* Servicio independiente desde inicio: Complejidad prematura para MVP
* Polling frecuente: Ineficiente pero más simple (opción de fallback)
* Solo notificaciones push: No funciona para usuarios en la web

**Consecuencias**

**Positivas:**

* Simplicidad en deployment y desarrollo inicial
* Transacciones de pujas simples (misma base de datos)
* Menos infraestructura que mantener
* Debugging más fácil

**Negativas:**

* Escalamiento limitado del componente de subastas
* WebSockets en Spring Boot menos performante que Node.js
* Todo el monolito debe escalar si solo subastas necesitan más recursos

**Plan de Migración Futura**

Cuando alcancemos los límites (>100 subastas concurrentes o latencia >500ms):

* Extraer servicio de subastas independiente (Node.js + Socket.io)
* Comunicación asíncrona vía mensajería (RabbitMQ/Kafka)
* Redis para gestión de estado de subastas activas

# ADR-007: Integración con Servicios Externos para Funcionalidades Críticas

**Estado:** Aceptado

**Fecha:** 2025-10-21

**Autores:** Angel Quishpe, Alan Rivera, Jorge Escobar

**Contexto**

Funcionalidades como procesamiento de pagos, certificación de autenticidad, logística y comunicaciones requieren expertise especializado y cumplimiento normativo. Desarrollarlas internamente sería costoso, riesgoso y distraería del core business.

**Decisión**

Integraremos servicios externos especializados:

* **Pagos:** Stripe/PayPal para procesamiento de tarjetas
* **Certificación:** APIs de blockchain para certificados de autenticidad
* **Logística:** APIs de DHL/FedEx para seguimiento de envíos
* **Comunicaciones:** SendGrid para emails, Twilio para SMS

**Consecuencias**

**Positivas:**

* Rápida implementación de features complejas
* Cumplimiento normativo garantizado por expertos
* Menor costo de desarrollo y mantenimiento
* Enfoque en core business (plataforma de arte)
* Servicios probados y confiables

**Negativas:**

* Dependencia de terceros
* Costos variables por transacción
* Menos control sobre la experiencia
* Riesgo de cambios en APIs externas
* Complejidad en testing (mocks necesarios)

# ADR-008: Estrategia de Despliegue con Contenedores Docker

**Estado:** Aceptado

**Fecha:** 2025-10-21

**Autores:** Angel Quishpe, Alan Rivera, Jorge Escobar

**Contexto**

Necesitamos consistencia entre entornos de desarrollo, staging y producción. Buscamos facilitar el escalamiento y despliegue de múltiples servicios (API principal, servicio de subastas, base de datos).

**Decisión**

Usaremos Docker para contenerización y Docker Compose para orquestación local. Para producción evaluaremos Kubernetes o servicios gestionados como AWS ECS.

**Alternativas Consideradas**

* Máquinas virtuales tradicionales: Pesadas, lentas de provisionar
* Despliegue directo en servidor: Inconsistencias entre entornos
* Serverless completo: No adecuado para WebSockets y aplicaciones stateful

**Consecuencias**

**Positivas:**

* Portabilidad entre entornos
* Aislamiento de dependencias
* Escalamiento horizontal simplificado
* CI/CD más eficiente
* Reproducibilidad de entornos

**Negativas:**

* Curva de aprendizaje inicial
* Overhead de recursos (mínimo)
* Complejidad adicional en debugging
* Requiere estrategia de gestión de logs centralizada