MK Music Archive — Technical Overview

Version: 1.0
Status: Public informational document
Date: 11 September 2026
Category: TECHNOLOGY / 04
Authority: Informational. This document explains the technical foundation for a public and stakeholder audience and does not replace the authoritative MK Music Registry technical specifications.



September 2026

TABLE OF CONTENTS

1. Why Technology Matters....3
2. Core Architecture....3
3. Technology Stack....3
4. The Registry Is Authoritative; the Public Web Is a Projection....4
5. Data, Identity, and History....4
6. Security by Design....4
7. Configuration and Environments....5
8. Testing and Quality Gates....5
9. Deployment, Operations, and Recoverability....6
10. Observability Without Exposing Sensitive Data....6
11. Maintainability and Future Scale....6
12. Engineering Discipline and AI-Assisted Development....7
13. What This Document Deliberately Does Not Publish....7
Document Governance....8

1. Why Technology Matters

MK Music Registry is intended as a long-lived cultural-memory system. Its technical architecture is therefore designed first for data integrity, traceability, security, maintainability, recoverability, and the ability to move the system to another operating environment.

The project deliberately avoids unnecessary infrastructure complexity. Instead of introducing multiple independent services simply because that is technically possible, the Registry starts as one well-structured system with explicit internal boundaries.

The guiding principle is simple: the simplest architecture that satisfies the cultural, security, and operational requirements is better than a more complex architecture without demonstrated need.

2. Core Architecture

The fundamental design and architectural decisions balance rapid initial delivery with a solid foundation that enables long-term sustainability and development.

MK Music Registry is being built as a modular monolith: one main Registry application and one transactional PostgreSQL database, with explicit internal modules and responsibilities.

The principal technical layers are:

- Domain — business and cultural concepts, entities, value objects, and invariants.
- Application — explicit use cases, orchestration, validation, and contracts between modules.
- Infrastructure — PostgreSQL/EF Core persistence, ASP.NET Core Identity, storage, integration adapters, and technical implementations.
- Web — the Blazor presentation layer and application composition.
- DatabaseMigrator — a controlled one-shot process for production database migrations.

Dependencies intentionally flow in one direction. Domain does not depend on the database, UI, or web framework. Application does not depend on Web. UI components do not access persistence directly.

3. Technology Stack

The main technology foundation is:

- NET 10 and C# 14
- ASP.NET Core
- Blazor Web App with Interactive Server
- Entity Framework Core
- PostgreSQL with Npgsql
- ASP.NET Core Identity
- MudBlazor for the back-office presentation layer
- Docker Compose for the production topology

- Caddy as the HTTPS reverse proxy
- GitHub as the source of code, technical documentation, and engineering history

The MVP deliberately does not introduce microservices, Kubernetes, a distributed message broker, Elasticsearch, or a separate workflow engine. PostgreSQL is the central transactional platform and, until measured needs prove otherwise, its search, trigram matching, indexing, and transactional capabilities are sufficient for the Registry.

4. The Registry Is Authoritative; the Public Web Is a Projection

The Registry is the source of structured, approved knowledge.

Public presentation does not read directly from internal Registry records. Instead, the system creates Public Projections — selected, public-safe representations of approved Registry knowledge. Approval and publication are separate steps.

The planned WordPress integration is a separate public presentation system. It can receive and present versioned Public Projections, but it cannot author Registry facts, approve records, or change internal research state.

This boundary allows the public system to evolve or be replaced without the Registry losing its authoritative role.

5. Data, Identity, and History

PostgreSQL stores the authoritative structured state. The Registry uses stable internal identities so the same subject remains traceable across corrections, new sources, and future integrations. External identifiers can assist matching but do not replace Registry identity.

History is not merely a technical audit add-on. Current editable state, revisions, Approved Snapshots, decisions, and audit events are intended to make it possible to explain not only what was accepted, but how the knowledge developed.

Files and evidence material are not treated as unstructured web-root assets. Access is mediated through a storage abstraction and rights/visibility rules.

6. Security by Design

The security model is deliberately conventional and testable.

Core principles include:

- deny by default;
- least privilege;

- server-side authorization;
- separation of human, machine, domain, and historical identities;
- private and restricted evidence does not become public merely because a Public Projection exists;
- standard framework controls are preferred over custom cryptography;
- secrets and sensitive data must not appear in source code, ordinary logs, or public diagnostics.

The back office uses ASP.NET Core Identity, policy-based authorization, and mandatory authenticator-app TOTP MFA for privileged users according to the approved security specification.

In Production, only the required web ports are publicly exposed through the reverse proxy. PostgreSQL, the Registry internal port, protected storage, and operational interfaces are not directly Internet-exposed.

7. Configuration and Environments

Exactly three application environments are approved:

• Development
• Testing
• Production

There is no permanent MVP Staging environment.

Configuration uses framework-native .NET mechanisms, typed Options, and startup validation. Secrets are never committed to Git, passed through command-line arguments, or placed in tests, logs, screenshots, or tickets. Development uses .NET User Secrets, while Production prefers key-per-file and externally supplied secrets.

Testing must remain completely isolated from Production resources.

8. Testing and Quality Gates

Quality is not demonstrated by a successful build alone.

Testing is layered:

- Domain tests verify pure domain rules and invariants.
- Application tests verify use cases, orchestration, authorization, and validation.
- Integration tests use real PostgreSQL through Testcontainers for persistence, migrations, concurrency, and database-specific behavior.
- Architecture tests protect dependency and layer boundaries.
- Web tests use bUnit for component behavior and Playwright for critical end-to-end journeys.
- Contract, security, accessibility, and operational verification are added where relevant.

The project uses a Fast Gate after meaningful changes and a Full Gate before substantial functionality is considered complete. Production readiness also requires operational evidence such as backup/restore and health verification.
SQLite and EF Core InMemory are deliberately not used as substitutes for PostgreSQL semantics.

9. Deployment, Operations, and Recoverability

Production is designed for provider-neutral Linux/VPS deployment.

The high-level topology is:

Internet → HTTPS reverse proxy → Registry application → PostgreSQL / protected storage

The Registry also communicates with the separate WordPress public system through a controlled HTTPS integration boundary.

Production uses Docker Compose, Caddy, a Registry container, PostgreSQL, protected persistent storage, persistent Data Protection keys, and external configuration/secrets.

Short controlled maintenance windows are acceptable for the MVP. Automatic failover and zero-downtime deployment are not requirements at this stage.

The practical MVP objective is an RPO (The maximum age of files that must be recovered from backup storage for normal operations to resume if a computer, system, or network goes down) of up to 24 hours and an RTO (Recovery Time Objective - The maximum tolerable duration of time that a business process or system can be down after a failure or disaster) within one business day after a recoverable single-VPS (Virtual Private Server) failure. These are operational objectives, not a high-availability guarantee.

A backup is not considered proven until it has been successfully restored.

10. Observability Without Exposing Sensitive Data

The Registry uses structured operational logs, health/readiness checks, and controlled operational diagnostics.

Logs should contain safe technical context such as correlation IDs, safe Registry IDs, projection versions, status, duration, and release/work state. They must not contain passwords, MFA secrets, Authorization headers, cookies, private Contributor data, restricted Evidence, protected file contents, or other secrets.

Monitoring focuses on material operational risks: Registry availability, PostgreSQL, disk capacity, backup health, stuck background work, publication failures, and relevant security conditions.

11. Maintainability and Future Scale

The architecture is optimized for a small team and realistic initial scale, but it is not designed as a disposable prototype.

The growth path is incremental:

1. increase VPS resources;
2. isolate PostgreSQL;
3. move files to object/NAS/institutional storage;
4. separate background workers when measured load justifies it;
5. add multiple Registry application instances only when real concurrency or availability needs justify them.

The same system can move to institutional Linux/on-premises infrastructure without redesigning the domain. New architectural complexity is introduced only when evidence shows a need.

12. Engineering Discipline and AI-Assisted Development

Code and technical documentation are organized in one GitHub monorepository.

The project favors explicit use cases and local mapping rather than additional abstraction frameworks by default. MediatR, AutoMapper, generic Repository, generic Unit of Work, BaseService, and BaseRepository are deliberately not approved as standard patterns.

This makes the system easier for both humans and AI coding agents to understand.

AI may assist implementation, testing, and documentation, but it operates within authoritative specifications, repository rules, and controlled quality gates. It must not independently invent new architectural layers, security models, or domain semantics.

13. What This Document Deliberately Does Not Publish

This Technical Overview is not an engineering runbook and does not replace the technical specifications.

It deliberately excludes:

- database schemas, columns, and indexes;
- detailed entity mappings;
- API endpoint catalogues or payload schemas;
- authentication secrets or credential-handling procedures;
- operationally detailed security topology;
- production commands, exact paths, or infrastructure credentials;
- step-by-step disaster-recovery procedures;
- the detailed repository implementation tree;

- authorization-policy internals;
- internal test fixtures and protected-data examples.

Its purpose is to demonstrate that MK Music Registry has a serious, maintainable engineering foundation without exposing detail that does not belong in a public publication.

Document Governance

This public document is derived from the approved Technical Architecture, Security Specification, Deployment/Operations/Disaster-Recovery Runbook, Solution & Repository Structure, Coding Standards, Test Strategy, and Environment/Configuration/Secrets-Management specifications.

Public authority: informational. If there is a difference, the approved numbered specifications remain the source of truth for architecture, security, operations, and implementation.

Update triggers: a material change to the application architecture, core technology stack, public-delivery boundary, security model, environment model, testing/quality gates, production topology, or recoverability targets.