KOOS Puzzle – Architecture Blueprint

Updated Draft v2 – Polished Edition

A complete, structured, and polished architectural summary for the next-generation KOOS Puzzle platform.

0. Architectural Ground Rules

These fundamental principles guide the entire system architecture. They define how pages, modules, and interactions should behave in order to maintain clarity, modularity, and long-term maintainability. **Single Responsibility Everywhere**
Each page or module handles exactly one purpose. This prevents complexity bloat and makes maintenance simpler. **No Cross■Page Coupling**
Even if multiple pages share similar Three.js code, duplication is preferred over coupling. Independence ensures robustness. **Every Movie Effect Gets Its Own Page**
Each effect—Turntable, Gravity, Explosion, Solve Reveal—exists as its own standalone entity, preventing massive, overloaded pages. **Canonicalization Is Sacred**
Hash calculations must be deterministic and isolated from metadata or UI data. **Avoid Premature Optimization**
Human-readable, stable code is prioritized over clever or compact code.

1. Unified Contract

The unified contract replaces the old dual■contract model and brings all puzzle shape, solution, and metadata into a single, structured document. This enables the system to treat a “shape” and a “solution” as two states of the same record, dramatically simplifying UGC, movies, sharing, and storage.

2. Hash Strategy

The architecture uses exactly two hashes: **Shape Hash** – a canonical fingerprint of the puzzle shape.
Solution Hash – a canonical fingerprint of the solution arrangement. A combined hash is not currently included in order to preserve simplicity.

3. Metadata Principles

Metadata is stored inside the unified contract but excluded from hash calculations. This enables infinite extensibility without breaking canonical identity. Examples include:
- Creator user ID
- Creation timestamps
- Country, school, or organization tags
- Competition status or award indicators
The metadata section may evolve significantly over time.

4. UGC Architecture (Movies + Images)

Movie generation is intentionally minimal on the server. Instead of storing video files: **Movie = Solution Record + Movie Parameter Record** The client uses these two components to generate animations dynamically. **Each movie effect is a separate page:**
- TurntableMoviePage
- GravityMoviePage
- ExplosionMoviePage
- SolveRevealMoviePage
- Any future effects (20+ possible)

This ensures modularity and prevents page bloat. **Sharing:**
Movies are shared via URLs that reconstruct them dynamically.
For platforms requiring uploads, users record locally generated video files.

5. Required Pages

The system is broken into clean, single■purpose pages: • Create Page – defines a shape
• Manual Solve Page – user-driven solving
• Automated Solve Page – system-driven solving
• Solution Viewer – visualize results
• Movie Pages – one per effect
• Gallery Page – hub for shapes, solutions, movies
• Purchase Page – buy physical puzzles
• Login/Registration Pages – identity management This separation dramatically simplifies maintenance and scaling.

6. Distribution Channels & Organizational Tagging

User accounts and solutions may include tags like: • Country
• State / Region
• City
• School / University
• Math or STEM organizations These tags support localized competitions, school brackets, trend analysis, and regional leaderboards.

7. Localization

The system uses a built■in localization framework, not Apple/Google auto■translation. This provides:
• Consistent terminology
• Custom phrasing for puzzle terminology
• Fully controlled multilingual expansion
• Identical behavior across all platforms

8. Platform Decision – Web Only

The decision to remain web■based is final. Three.js already delivers exceptional performance, a smooth experience, and global accessibility without app store bottlenecks. This architecture is built for a web-first future.

9. Modes of Operation

There are three operational modes: **Development Mode**
Debugging, diagnostics, experimental tools, internal features. **User Mode (Production)**
Clean UI, optimized experience, no debug tooling. **Beta Mode**
Limited rollouts, A/B testing, early adopters, feature previews.

10. Scalability

The core backend uses Supabase (Postgres + Auth + Storage) running on AWS■scale infrastructure. The architecture is intentionally lightweight: • JSON contracts
• Minimal stored media
• On■device rendering of movies
• Stateless endpoints
All of this supports viral scaling to millions of users with minimal friction.

11. Testing & Quality Assurance

The next implementation cycle emphasizes robust automated testing: • Contract canonicalization tests
• Solver determinism tests
• UI regression tests
• Cross■browser tests
• Performance tests
Development Mode exposes debugging tools for visualizing lattice edges, paths, physics behavior, etc.

12. Open Future Topics

Topics to address after core launch: • Real■time multi■player solving
• NFT / tokenized shapes or solutions
• UGC moderation for extreme parameters
• Curated thematic galleries
• Creator achievement systems

- Global live events