ARCHITECTURE CASE STUDY 7C / 01

Autonomous TangoKunstWerk workbench

A WordPress-based engineering system for course operations, safe previews and evidence-led quality control.

Context and objective

Recurring class schedules and absences needed to be managed consistently across several locations. Separate interfaces or duplicated data would create risks of conflicting course dates and unintended publication.

The goal was one authoritative data source, accessible mobile administration and a strict distinction between preview, approval and publishing.

Solution architecture

WordPress holds the production data. Modern and Classic provide two interfaces to the same course options. The calendar assistant validates date ranges, previews affected sessions and requires explicit approval for publishing. PowerShell gateways and dedicated MCP/browser tools support technical orchestration and verification.

Saarbrücken, Speyer and Bad Dürkheim are in scope. Homburg is deliberately excluded from automated scheduling changes.

COMPONENT AND RESPONSIBILITY FLOW

Input dates	Validation	Real WP preview	Explicit approval

Design principle

Separate decision rights, execution and acceptance. The solution is intentionally bounded: protect existing systems and make authority auditable rather than assume unlimited autonomy without evidence.

Implementation and integration

Selected technical steps, interfaces and design choices without exposing protected implementation details.

Implementation approach

The UI accepts multiple unambiguous date formats, rejects invalid values and allows controlled location selection. The integration avoids a duplicate course database and competing writers.

Integration checks against the deployed PHP bundle exposed a missing comparison class; the missing component was added exactly once. This was a meaningful discovery beyond merely loading the admin interface successfully.

Quality assurance and acceptance gates

Evidence includes 26 PHP date tests, 18 browser cases and 28+11 checks against the actual code. A real authorized WordPress preview POST produced one proposed change with zero publish POSTs; five course options stayed unchanged.

Modern/Classic passed 14/14 read-only live admin views. The owner personally accepted the calendar assistant in iPhone Safari. Another 21/21 isolated HTTP controller tests with WordPress stubs covered preview, one-time token handling, publish and restore.

Architectural rationale

The solution supports incremental change and small, observable delivery steps. Each gate has a concrete acceptance condition. Failures and limitations remain visible instead of being concealed behind a green status label.

Verification strategy

- Record a baseline and limit the change scope.
- Check the integration path actually in use, not just an isolated component.
- Exercise negative cases and explicitly forbidden states.
- Preserve findings, residual uncertainty and reproducible handover records.

Evidence, risk and learning

A credible reference separates actual observations, isolated tests and operational objectives that remain open.

Security boundaries and remaining limitations

Publishing is gated by a concrete approval; a preview does not constitute a release. Negative tests, hash readback, version/log checks and protection of unaffected options underpin the workflow.

The first real publication of an absence with production readback is still pending. The isolated restore tests are not equivalent to a full WordPress server restore.

Impact and personal contribution

The case combines UX, backend state, security and test-driven integration. A real bundle defect was found, corrected and verified rather than hidden behind a green-looking interface.

My role covered requirements, course policies, usability acceptance, implementation coordination, authorization gates and independent QA; software implementation was AI-assisted.

Transferability

The pattern can inform AI integration, internal automation and developer-experience tooling: explicit interfaces, least privilege, documented state, cost control and reproducible verification. Applying it elsewhere requires system-specific security and acceptance work.

Lessons learned

Meaningful architecture evidence describes more than an end state. It explains how defects were discovered, what a test can and cannot establish, and who has authority over production release. Documentation therefore forms part of the engineering deliverable rather than a marketing afterthought.

Evidence basis: internal project records dated 22–25 November 2026, test reports and portfolio register. This application edition intentionally excludes private paths, tokens, repository URLs and personal system information.