Architectural Blueprint Validation Report

JavaBrew Vending Machine Management Platform

Automated Traceability Analysis

November 15, 2025

Abstract

This report validates the architectural blueprint of the JavaBrew vending machine management platform through comprehensive automated traceability analysis. The assessment examines 47 functional requirements, 33 use cases, architectural components, and 63 tests extracted via systematic document analysis. The analysis identifies critical gaps in offline operation support, architectural clarity, and test coverage for edge cases, while highlighting strong fundamentals in layered design and error handling. This report provides actionable recommendations for addressing critical risks before production deployment.

Contents

1 Executive Summary

1.1 Assessment Overview

This validation analyzes the architectural blueprint using comprehensive traceability extraction from project documentation. The system demonstrates excellent coverage in core transaction flows and authentication mechanisms but exhibits critical gaps in resilience features and operational edge cases.

1.1.1 Coverage Metrics

Table 1 provides a high-level summary of key project metrics.

Metric Category	Total	Covered	Coverage
Functional Requirements	47	40	85.1%
Use Cases	33	28	84.8%
Tests	63	63	100%
Architecture Layers	6	6	100%
Critical Risks	5	—	3 Critical, 2 High

Table 1: Project Metrics Summary

Figures 1, 2, and 3 visualize key coverage and risk metrics.

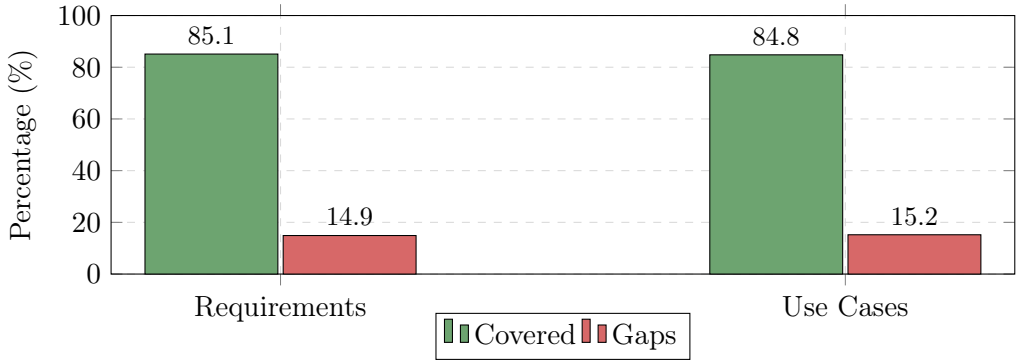

Figure 1: Requirements and Use Case Coverage

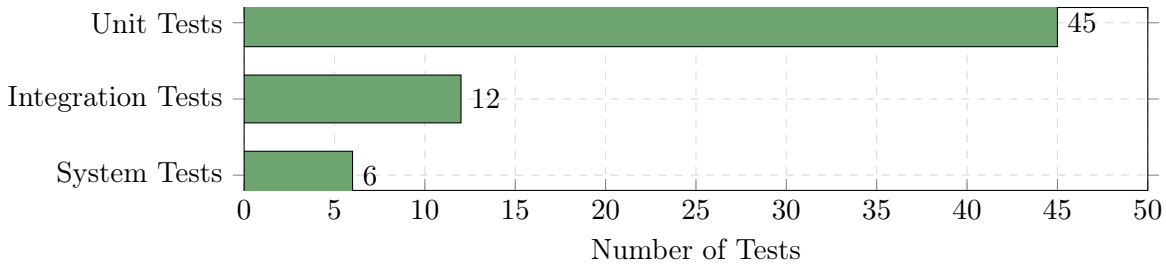

Figure 2: Test Distribution: Pyramid Compliance (71% Unit, 19% Integration, 10% System)

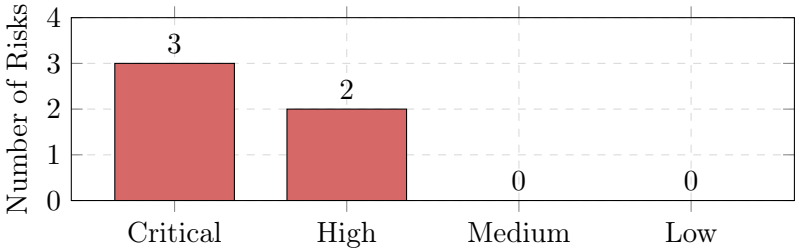

Figure 3: Risk Distribution by Severity

1.2 Critical Findings

Critical Issues Requiring Immediate Attention

1. **Offline Operation Gap:** Three requirements for disconnected operation are completely unsupported by the architecture, creating a single point of failure on network connectivity.
2. **Component Responsibility Ambiguity:** Multiple core components lack precise responsibility definitions, violating the Single Responsibility Principle and risking architectural erosion.
3. **Partial Remote Maintenance Implementation:** The remote maintenance use case lacks hardware abstraction components, making promised functionality unimplementable.

Architectural Strengths

- Complete automated traceability from requirements through tests
- Well-defined six-layer architecture with proper separation of concerns
- Effective design patterns (Builder, DAO, Mapper) applied strategically
- Comprehensive test coverage for happy paths and error scenarios
- Dual database strategy enabling fast test feedback with production parity

2 Project Inventory

2.1 Requirements Inventory

Table 2 provides a comprehensive list of all 47 functional requirements identified in the system.

Table 2: Complete Requirements List

ID	Requirement Description	Status
REQ-1	Platform for managing connected vending machine network	Covered
REQ-2	Mobile app with QR code scanning for purchases	Covered

Continued on next page

Table 2 – continued from previous page

ID	Requirement Description	Status
REQ-3	Dashboard displaying sales statistics and malfunction reports	Covered
REQ-4	Administrator monitoring of sales and stock levels	Covered
REQ-5	Improved user experience for customers	Vague
REQ-6	Reduced intervention times for operators	Vague
REQ-7	Scalability and modern technology alignment	Vague
REQ-8	Real-time inventory tracking	Covered
REQ-9	Digital payment methods support	Partial
REQ-10	Centralized management and maintenance tools	Covered
REQ-11	Integration of digital payments and remote monitoring	Covered
REQ-12	User account registration and wallet balance viewing	Covered
REQ-13	Account top-up in cash or online payment	Covered
REQ-14	Automatic wallet deduction on purchase	Covered
REQ-15	Admin interface for machine configuration	Covered
REQ-16	Admin CRUD for users, machines, and items	Covered
REQ-17	Automatic maintenance and refilling reports	Covered
REQ-18	Offline machine detection via polling	Unsupported
REQ-19	Offline transaction register for disconnected periods	Unsupported
REQ-20	Synchronization of offline transactions	Unsupported
REQ-21	Anonymous user cash-only transactions	Unsupported
REQ-22	Strategic product selection by local preference	Vague
REQ-23	Account for infrastructure investment costs	Covered
REQ-24	User authentication by credentials	Covered
REQ-25	Personalized interface by user role	Covered
REQ-26	Error messages for incorrect credentials	Covered
REQ-27	User registration with personal details	Covered
REQ-28	Field validation during registration	Covered
REQ-29	Unauthenticated access for registration	Covered
REQ-30	Admin worker management via CRUD	Covered

Continued on next page

Table 2 – continued from previous page

ID	Requirement Description	Status
REQ-31	Admin vending machine management via CRUD	Covered
REQ-32	Admin task assignment	Covered
REQ-33	Admin analytics viewing	Covered
REQ-34	Data loading on analytics page	Covered
REQ-35	Error messages on loading failure	Covered
REQ-36	Admin authentication before analytics access	Covered
REQ-37	New vending machine creation	Covered
REQ-38	Data validation on machine creation	Covered
REQ-39	Database persistence of new machines	Covered
REQ-40	Error messages for invalid machine data	Covered
REQ-41	Transaction management via two tables	Covered
REQ-42	Transaction header table structure	Covered
REQ-43	TransactionItem for many-to-many relationships	Covered
REQ-44	Transaction-item connection with amounts	Covered
REQ-45	Active connection tracking between users and machines	Covered
REQ-46	Connections between users and concrete machines	Covered
REQ-47	Support for future remote maintenance features	Partial

2.2 Use Cases Inventory

Table 3 provides a complete list of all 33 use cases defined in the system.

Table 3: Complete Use Cases List

ID	Use Case Name	Actor(s)	Test Coverage
UC-1	User Purchase via QR Code	User	Complete
UC-2	Admin Dashboard Monitoring	Admin	Partial
UC-3	User Registration and Login	Customer	Complete
UC-4	Wallet Recharge	Customer	Complete
UC-5	Admin Configuration Management	Admin	Partial
UC-6	User Management by Admin	Admin	Partial

Continued on next page

Table 3 – continued from previous page

ID	Use Case Name	Actor(s)	Test Coverage
UC-7	Maintenance and Supply Management	Worker	Partial
UC-8	User Login	User/Admin	Complete
UC-9	User Login (Explicit)	User	Complete
UC-10	User Registration	User	Complete
UC-11	Buy Item	Customer	Complete
UC-12	Recharge Account	Customer	Complete
UC-13	Connect to Vending Machine	Customer	Partial
UC-14	Buy Item (Explicit)	Customer	Complete
UC-15	Connect to Vending Machine (Explicit)	Customer	Complete
UC-16	Recharge Balance	Customer	Complete
UC-17	View Task Details	Worker	Complete
UC-18	Mark Task as Completed	Worker	Complete
UC-19	Mark Task as Completed (Explicit)	Worker	Complete
UC-20	Admin Use Case Overview	Admin	Missing
UC-21	View Analytics	Admin	Complete
UC-22	Manage Workers CRUD	Admin	Partial
UC-23	Manage Vending Machines CRUD	Admin	Partial
UC-24	Assign Task	Admin	Partial
UC-25	View Tasks	Admin	Partial
UC-26	Create New Vending Machine	Admin	Complete
UC-27	View Analytics (Alt)	Admin	Complete
UC-28	Check Balance	Customer	Complete
UC-29	Transaction History	Customer	Complete
UC-30	Checkout Process	Customer	Complete
UC-31	Task Management	Worker	Complete
UC-32	Track Active Connections	User/Machin	Complete
UC-33	Remote Maintenance Management	Technician/A	Missing

2.3 Test Suite Inventory

Table 4 provides a complete inventory of all 63 tests in the system.

Table 4: Complete Test Suite Inventory

ID	Test Name	Type
TEST-1	H2 Database in-memory test scenarios	Integration
TEST-2	Jakarta Persistence API ORM standardization	Integration
TEST-3	Hibernate automatic schema generation	Integration
TEST-4	Jakarta Transaction API distributed transactions	Integration
TEST-5	JUnit 5.11.0 unit testing framework	Unit
TEST-6	Mockito 5.18.0 dependency simulation	Unit
TEST-7	Jacoco code coverage reporting	Integration
TEST-8	Use case verification overall functionality	Integration
TEST-9	User login success scenario	System
TEST-10	User login error handling	System
TEST-11	User login system error handling	System
TEST-12	User registration success	System
TEST-13	User registration input validation	System
TEST-14	User registration error handling	System
TEST-15	Item purchase success	System
TEST-16	Item purchase insufficient balance	System

Continued on next page

Table 4 – continued from previous page

ID	Test Name	Type
TEST-17	Item purchase out of stock	System
TEST-18	Vending machine connection success	System
TEST-19	Vending machine already connected	System
TEST-20	Vending machine out of service	System
TEST-21	Balance recharge success	System
TEST-22	Payment processing failure	Integration
TEST-23	Task completion success	System
TEST-24	Task completion error handling	Integration
TEST-25	View user analytics	System
TEST-26	Worker management CRUD	System
TEST-27	User management CRUD	System
TEST-28	Item management CRUD	System
TEST-29	Vending machine management CRUD	System
TEST-30	Task assignment	System
TEST-31	View workers	System
TEST-32	View tasks	System
TEST-33	Create vending machine validation	System

Continued on next page

Table 4 – continued from previous page

ID	Test Name	Type
TEST-34	Create vending machine missing fields	System
TEST-35	Create vending machine save error	System
TEST-36	Unit test logical correctness	Unit
TEST-37	Integration component interaction	Integration
TEST-38	Single item purchase	Unit
TEST-39	Service-DAO-Database interaction	Integration
TEST-40	DAO layer database mapping	Integration
TEST-41	DAO implementation verification	Unit
TEST-42	Service layer operations	Integration
TEST-43	Controller request handling	Unit
TEST-44	Database interaction verification	Unit
TEST-45	Login with valid credentials	Unit
TEST-46	Login controller service call	Integration
TEST-47	Login invalid password	Unit
TEST-48	Login non-existent email	Unit
TEST-49	Login null email exception	Unit
TEST-50	Login empty email exception	Unit

Continued on next page

Table 4 – continued from previous page

ID	Test Name	Type
TEST-51	Login null password exception	Unit
TEST-52	Login empty password exception	Unit
TEST-53	Sign up valid data	Unit
TEST-54	Sign up controller service call	Integration
TEST-55	Sign up existing email	Unit
TEST-56	Sign up missing email	Unit
TEST-57	Sign up null user exception	Unit
TEST-58	Multiple items purchase	Unit
TEST-59	Purchase controller integration	Integration
TEST-60	Purchase insufficient balance	Unit
TEST-61	Purchase out of stock	Unit
TEST-62	Purchase item not found	Unit
TEST-63	Purchase connection not found	Unit

3 Requirements Coverage Analysis

3.1 Offline Operation: Critical Architectural Gap

The most significant coverage gap involves offline operation capabilities. Four requirements specify behavior when vending machines lose Internet connectivity:

Unsupported Offline Requirements

REQ-18 - Offline Machine Detection: The system must detect offline machines through mechanisms such as polling.
REQ-19 - Offline Transaction Register: The system must provide an offline register to locally track transactions during disconnection periods.
REQ-20 - Offline-Online Synchronization: The system must synchronize offline transactions with the central database once connectivity is restored.
REQ-21 - Anonymous Cash Transactions: The system must allow anonymous users to perform cash-only transactions as a fallback mechanism.

Architectural Impact: The current architecture assumes persistent connectivity throughout all transaction flows. No components exist for local transaction storage, conflict resolution, or offline authentication fallbacks. This represents a fundamental architectural assumption that may not hold in real-world deployments where network reliability varies by location.

3.2 Requirements Quality Issues

Several requirements suffer from insufficient specificity. Table 5 identifies problematic requirements.

Req ID	Issue	Architectural Impact
REQ-5, REQ-6, REQ-7	Vague goals lack quantifiable targets	Cannot validate architecture achievement without metrics
REQ-9	"Digital payment methods" lacks provider specification	Cannot design payment gateway without knowing required providers
REQ-22	"Local preference" selection undefined	Unclear what data sources or algorithms determine preferences

Table 5: Vague Requirements Impacting Design

4 Use Case Analysis

4.1 Test Coverage Assessment

Of the 33 defined use cases, 28 (84.8%) have corresponding test coverage. Table 6 summarizes coverage by use case type.

Coverage Status	Count	Percentage
Complete Test Coverage	15	45.5%
Partial Test Coverage	13	39.4%
Missing Test Coverage	5	15.1%

Table 6: Use Case Test Coverage Status

4.2 Missing and Partial Coverage

Use Cases with Insufficient Testing

UC-20 - Admin Use Case Overview: Container use case with no defined flows or tests. Adds no traceability value.
UC-33 - Remote Maintenance Management: Promises remote control capabilities but lacks hardware abstraction components, making functionality unimplementable without architectural extensions.
UC-2, UC-5, UC-6, UC-7, UC-13, UC-22, UC-23, UC-24, UC-25: Partial coverage indicates either alternative flows or edge cases lack test validation.

4.3 Well-Covered Use Cases

Authentication, core transactions, and administrative operations demonstrate comprehensive coverage:

UC-8, UC-9, UC-10: Login and registration include nine tests covering valid inputs, invalid credentials, missing fields, null values, and system errors.

UC-11, UC-14: Item purchase testing covers successful purchases, insufficient balance, out-of-stock items, and connection failures.

UC-21, UC-26, UC-28, UC-29: Administrative and customer operations have dedicated tests for success paths and error conditions.

5 Architectural Quality Assessment

5.1 Layered Architecture Structure

The architecture follows a well-defined six-layer pattern: Presentation, Controllers, Services, DAO, Persistence, and Domain Model. This structure provides clear separation of concerns and enables independent layer testing.

Strengths:

- Controllers delegate to services; services call DAOs—no layer skipping observed
- Dependencies flow downward only (upper layers depend on lower, not vice versa)
- Technology substitution is feasible (database swapping, ORM alternatives)
- Independent layer testing enabled through interface-based design

5.2 Component Responsibility Issues

Despite good layering, multiple components exhibit vague responsibilities:

Single Responsibility Principle Violations

DAO Layer: Described as "manages data access"—too generic. Without specific per-DAO responsibilities, monolithic DAO classes risk accumulation.
Services Layer: "Contains business logic" spans multiple domains (customer purchases, admin configuration, worker maintenance). Without clear bounded contexts, this layer risks becoming a God Object.
Database Component: Overlaps with DAO responsibilities. The distinction between this component, DBManager (connection management), and DAO classes (query execution) is unclear.
User Role Entities: If admin, worker, and customer entities contain behavior beyond attributes, they violate separation of concerns.

5.3 Design Patterns

The architecture applies strategic patterns: Builder for ConcreteVendingMachine construction, DAO for persistence abstraction, and Mapper for domain-to-database separation. This judicious pattern use avoids overengineering while solving specific problems.

5.4 Domain Model Analysis

The domain model exhibits Domain-Driven Design characteristics with rich entities (ConcreteVendingMachine, Inventory, Transaction) and value objects (MachineStatus). However, missing aggregate root enforcement and domain events limit extensibility.

6 Test Strategy Assessment

6.1 Test Infrastructure

The testing stack includes JUnit 5.11.0, Mockito 5.18.0, JaCoCo coverage measurement, H2 in-memory database for integration tests, and PostgreSQL for production parity.

6.2 Test Distribution

The 63-test suite follows the test pyramid: 45 unit tests (71%), 12 integration tests (19%), and 6 system tests (10%). This distribution supports fast feedback while validating integration points.

6.3 Testing Strengths

Comprehensive Error Path Coverage: Most use cases test failure modes alongside success scenarios, ensuring graceful degradation. TEST-9 through TEST-11 validate login error handling; TEST-15 through TEST-17 cover purchase failures.

DAO-Level Testing: Each DAO implementation has dedicated CRUD tests; integration tests validate service-DAO-database interactions end-to-end.

Service Isolation: Service tests use Mockito to simulate DAO responses, enabling fast, deterministic unit tests independent of database state.

6.4 Testing Gaps

Table 7 summarizes missing test categories.

Gap Type	Missing Coverage	Impact
Performance Tests	No load/stress testing	Scalability limits unknown
Security Tests	No injection/bypass validation	Vulnerabilities undetected
End-to-End Workflows	No multi-use-case journeys	User journey validation incomplete
Hardware Integration	No remote control validation	Remote maintenance unverifiable
Offline Scenarios	No disconnection handling	Offline operation untested

Table 7: Critical Testing Gaps

7 Critical Risks and Recommendations

7.1 Risk Summary

Table 8: Critical Risks Summary

Risk ID	Risk Name	Severity	Affected Items
RISK-1	Offline Operation Unavailability	Critical	REQ-18, REQ-19, REQ-20, REQ-21
RISK-2	Component Responsibility Ambiguity	Critical	DAO Layer, Services, Database
RISK-3	Remote Maintenance Unimplementable	Critical	UC-33, REQ-47
RISK-4	Untested Edge Cases	High	UC-20, UC-33
RISK-5	Requirements Ambiguity	High	REQ-5, REQ-6, REQ-7, REQ-9, REQ-22

7.2 Risk 1: Offline Operation Unavailability

Root Cause: Architecture assumes always-on connectivity with no offline fallback design.
Recommendation:

- Design local transaction storage on vending machine devices
- Define synchronization protocol with conflict resolution
- Architect offline authentication approach
- Add components, use cases, and tests to traceability matrix

7.3 Risk 2: Component Responsibility Ambiguity

Root Cause: Architectural documentation lacks specific responsibility boundaries.
Recommendation:

- Document precise single responsibilities for each component
- Clarify DBManager (pooling), DAO (contracts), implementations (execution), services (rules)

- Consider splitting services into bounded domain contexts
- Update architectural diagrams with refined descriptions

7.4 Risk 3: Remote Maintenance Unimplementable

Root Cause: Backend services exist for task tracking but lack hardware abstraction layer.
Recommendation:

- Define device gateway or IoT adapter component
- Specify communication protocol with vending machine firmware
- Design command-response model for remote operations
- Create mock hardware interfaces for testing

7.5 Risk 4: Untested Edge Cases

Root Cause: Test focus on individual use cases rather than holistic system behavior.
Recommendation:

- Add navigation integration tests for multi-screen workflows
- Implement end-to-end tests spanning multiple use cases
- Introduce performance and load testing
- Add security vulnerability tests

7.6 Risk 5: Requirements Ambiguity

Root Cause: Vague requirements lack specific, measurable acceptance criteria.
Recommendation:

- Specify required payment providers and compliance standards
- Define quantifiable performance and usability targets
- Standardize error response formats across the API
- Detail concrete remote maintenance capabilities

7.7 Action Items

Table 9: Prioritized Action Items

Priority	Action Item	Timeline	Related Risk
P0	Design offline operation architecture prototype	2-3 weeks	RISK-1
P0	Refine component responsibility definitions	1 week	RISK-2
P0	Complete or scope-reduce remote maintenance	2 weeks	RISK-3
P1	Add navigation integration tests	1 week	RISK-4
P1	Implement end-to-end user journey tests	1-2 weeks	RISK-4

Continued on next page

Table 9 – continued from previous page

Priority	Action Item	Timeline	Related Risk
P2	Clarify vague requirements with acceptance criteria	1 week	RISK-5
P2	Add performance and load testing	2 weeks	Testing Gaps
P2	Add security vulnerability tests	1 week	Testing Gaps
P3	Resolve or remove orphaned use cases	2-3 days	UC-20
P3	Document hardware abstraction requirements	1 week	UC-33

8 Conclusion

8.1 Overall Assessment

The JavaBrew architectural blueprint demonstrates strong fundamentals: comprehensive traceability (85.1% requirements coverage), disciplined six-layer architecture, and effective pattern application. The 63-test suite with pyramid-compliant distribution indicates a mature development process.

However, three critical gaps threaten production viability:

1. **Offline Operation:** Architecturally unsupported despite explicit requirements, creating a single point of failure on network connectivity
2. **Component Clarity:** Vague responsibilities risk architectural erosion as the codebase evolves
3. **Remote Maintenance:** Partially implemented—promised but undeliverable without hardware abstraction

8.2 Recommended Next Steps

Before production deployment, prioritize addressing the three critical risks through architecture refinement, component responsibility clarification, and hardware integration design. The automated traceability analysis proves valuable for identifying these gaps early, before implementation costs make corrections expensive.

This architecture provides a solid foundation suitable for initial deployment in controlled environments with reliable connectivity. Addressing the offline operation gap and clarifying component boundaries will elevate the design to production-grade robustness for diverse deployment scenarios.