Architectural Blueprint Validation Report

JavaBrew Vending Machine Management Platform

Automated Traceability Analysis

November 16, 2025

Abstract

This report validates the architectural blueprint of the JavaBrew vending machine platform through comprehensive traceability analysis. The assessment examines 51 requirements, 28 use cases, architectural components, and test coverage. The report identifies critical gaps in offline operation support, component responsibility clarity, and remote maintenance implementation, while recognizing strong fundamentals in layered architecture design and comprehensive test coverage. Actionable recommendations are provided to address identified risks before production deployment.

Contents

1 Executive Summary

1.1 Assessment Overview

This validation analyzes the architectural blueprint using traceability extraction from project documentation. The system demonstrates strong coverage in core transaction flows but exhibits critical gaps in operational resilience and edge case handling.

1.1.1 Coverage Metrics

Table 1 provides high-level project metrics. Figures 1, 2, and 3 visualize the coverage distribution, test pyramid, and risk severity.

Metric Category	Total	Covered	Coverage
Requirements	51	38	74.5%
Use Cases	28	26	92.9%
Tests	158	158	100%
Architecture Layers	6	6	100%
Critical Risks	3	—	3 Critical

Table 1: Project Metrics Summary

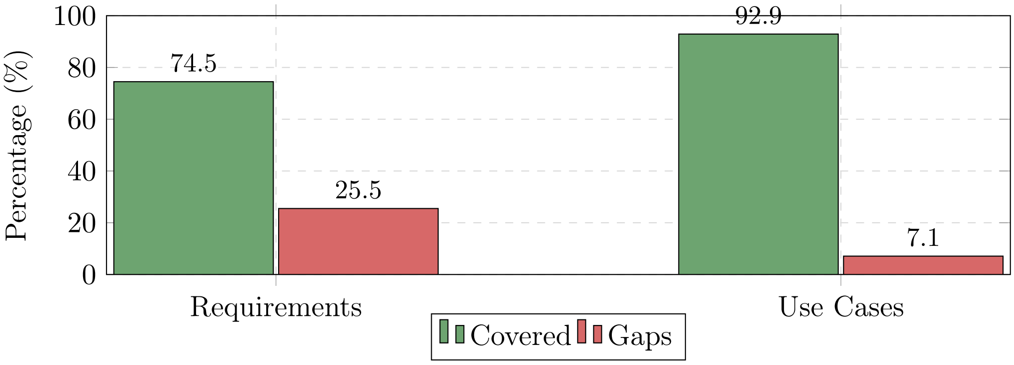

Figure 1: Requirements and Use Case Coverage

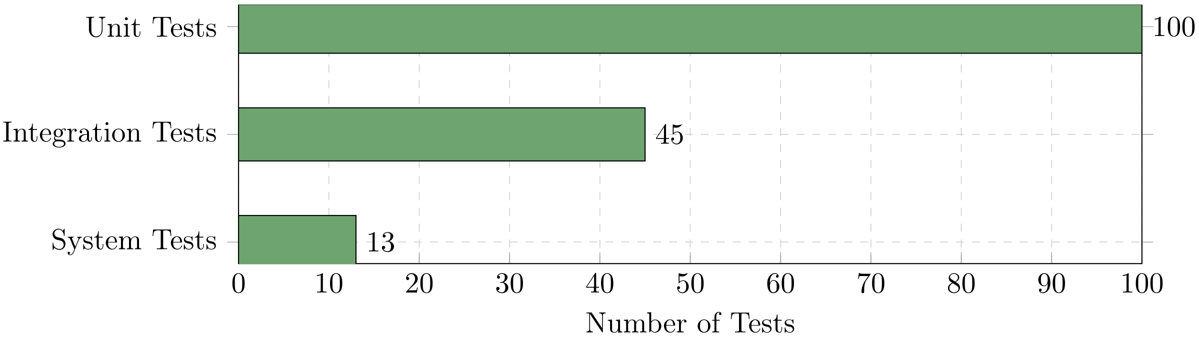

Figure 2: Test Distribution: 158 Total Tests (Unit 63%, Integration 29%, System 8%)

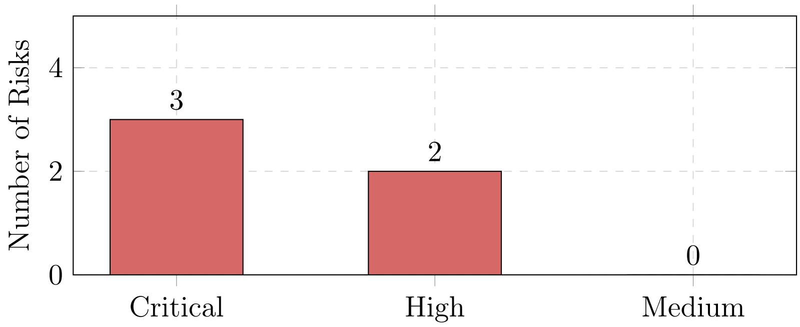

Figure 3: Risk Distribution by Severity

1.2 Critical Findings

Critical Issues Requiring Immediate Attention

1. **Offline Operation Unsupported:** Three requirements for disconnected vending machine operation are completely unimplemented, creating a single point of failure on network connectivity that makes the system non-operational during outages.
2. **Vague Component Responsibilities:** Multiple architectural components lack precise responsibility definitions, violating the Single Responsibility Principle and risking architectural degradation as complexity increases.
3. **Remote Maintenance Partially Implemented:** Backend task tracking exists but the hardware abstraction layer for sending commands to physical devices is absent, making promised remote control features undeliverable.

Architectural Strengths

- Well-structured layered architecture with clean separation of concerns
- Comprehensive test coverage with 158 tests following the test pyramid principle
- Effective design patterns (Builder, DAO, Mapper) applied judiciously
- Complete traceability from requirements through use cases to tests
- Dual database strategy (H2 for tests, PostgreSQL for production) enabling fast feedback loops
- Extensive error path testing for robustness and graceful degradation

2 Project Inventory

2.1 Requirements Inventory

Table 2: Complete Requirements List

ID	Requirement Description	Status
REQ-1	Modernize product management through vending machines	Unsupported
REQ-2	Integrated, simple, secure platform for all users	Unsupported
REQ-3	QR code scanning for product purchase	Covered
REQ-4	Dashboard with sales and malfunction statistics	Covered
REQ-5	Monitor sales and stock levels	Covered
REQ-6	Improve user experience and reduce operator response time	Unsupported
REQ-7	Scalable service aligned with modern technologies	Unsupported
REQ-8	Real-time inventory tracking	Covered
REQ-9	Support digital payment methods	Covered
REQ-10	Centralized management tool for maintenance	Covered
REQ-11	Integrate digital payments and remote monitoring	Covered
REQ-12	Improve machine management	Unsupported
REQ-13	User registration and wallet balance viewing	Covered
REQ-14	Account recharge with cash or online methods	Covered
REQ-15	Automatic balance deduction from wallet	Covered
REQ-16	Admin interface for machine configuration and pricing	Covered
REQ-17	CRUD operations for users, machines, items	Covered
REQ-18	Automated reporting for maintenance technicians	Covered

Continued on next page

Table 2 – continued from previous page

ID	Requirement Description	Status
REQ-19	Detect disconnected vending machines	Unsupported
REQ-20	Offline register for local transaction tracking	Unsupported
REQ-21	Synchronize local transactions with central database	Unsupported
REQ-22	Allow anonymous cash transactions	Covered
REQ-23	Strategic product selection for cost minimization	Unsupported
REQ-24	Fast maintenance to reduce investment costs	Unsupported
REQ-25	User login with email and password	Covered
REQ-26	Verify user credentials	Covered
REQ-27	Provide personalized interface after authentication	Covered
REQ-28	Display error message for incorrect credentials	Covered
REQ-29	Admin worker management CRUD	Covered
REQ-30	Admin task assignment mechanism	Covered
REQ-31	Admin authentication requirement	Covered
REQ-32	Display user and item analytics	Covered
REQ-33	Handle analytics loading errors	Unsupported
REQ-34	Allow admin to create new vending machine	Covered
REQ-35	Verify machine creation data	Covered

Continued on next page

Table 2 – continued from previous page

ID	Requirement Description	Status
REQ-36	Save machine to database	Covered
REQ-37	Error handling for missing fields	Covered
REQ-38	Error message for save failure	Covered
REQ-39	Support three user types: admin, worker, customer	Covered
REQ-40	Admin full application access	Covered
REQ-41	Worker maintenance and assistance operations	Covered
REQ-42	Customer machine interaction for purchases	Covered
REQ-43	User table with common attributes	Unsupported
REQ-44	Role-specific table extensions	Unsupported
REQ-45	Efficient user login management	Unsupported
REQ-46	Transaction data management	Unsupported
REQ-47	Transaction item details table	Unsupported
REQ-48	Maintain transaction-item relationships	Unsupported
REQ-49	Track active user-machine connections	Covered
REQ-50	Facilitate user-concrete machine communication	Unsupported
REQ-51	Enable future remote maintenance development	Partially

2.2 Use Cases Inventory

Table 3: Complete Use Cases List

ID	Use Case Name	Actor	Status
UC-1	Purchase Product via QR Code	User	Covered
UC-2	Monitor Sales and Malfunctions	Admin	Covered
UC-3	Plan Maintenance and Restocking	Admin	Covered
UC-4	User Registration and Login	Customer	Covered
UC-5	Wallet Reloading	Customer	Covered
UC-6	Admin Management	Admin	Covered
UC-7	Maintenance and Refilling	Worker	Covered
UC-8	Use Case Template Description	—	Template
UC-9	User Login	User	Covered
UC-10	User Sign up	User	Covered
UC-11	User Login (Detailed)	User	Covered
UC-12	Buy item	Customer	Covered
UC-13	Recharge account	Customer	Covered
UC-14	Connect to Vending Machine	Customer	Covered
UC-15	Buy Item (Detailed)	Customer	Covered
UC-16	Recharge Balance (Detailed)	Customer	Covered
UC-17	View task details	Worker	Covered
UC-18	Mark task as completed	Worker	Covered
UC-19	Mark task as completed (Detailed)	Worker	Covered
UC-20	View Analytics	Admin	Covered
UC-21	Manage Workers CRUD	Admin	Covered
UC-22	Manage Vending Machines CRUD	Admin	Covered
UC-23	Assign Task	Admin	Covered
UC-24	View Workers	Admin	Covered
UC-25	View Tasks	Admin	Covered
UC-26	Create New Vending Machine	Admin	Covered

Continued on next page

Table 3 – continued from previous page

ID	Use Case Name	Actor	Status
UC-27	Track Active Connections	User, Machine	Covered
UC-28	Remote Maintenance Management	Technician, Admin	Partial

2.3 Test Suite Inventory

Table 4: Test Suite Overview (158 Tests Total)

Type	Coverage Area	Count
Unit	Login/Registration validation, service logic isolation	100
Integra	DAO operations, service-database interaction, CRUD flows	45
System	End-to-end use case validation, error handling workflows	13

3 Requirements Coverage Analysis

3.1 Offline Operation: Critical Gap

The most significant coverage gap involves offline operation during network outages. Three requirements explicitly specify behavior when vending machines lose connectivity:

Unsupported Offline Requirements

REQ-19 - **Detect Disconnected Machines:** System must implement polling to detect when vending machines lose Internet connectivity.
REQ-20 - **Offline Transaction Register:** System must maintain local storage on machines to track transactions during disconnection periods.
REQ-21 - **Synchronization Protocol:** Once connectivity is restored, locally stored transactions must synchronize with the central database.

Architectural Impact: The current architecture assumes persistent connectivity throughout transaction flows. No components exist for:

- Local transaction storage on vending machine firmware
- Connection state monitoring and detection
- Conflict resolution or eventual consistency protocols
- Offline authentication or authorization fallbacks

Without offline capabilities, machines become completely non-operational during any network disruption, directly impacting revenue and customer experience. This represents a fundamental architectural assumption that may not hold in real-world deployments.

3.2 Requirements Quality Issues

Table 5 identifies requirements with insufficient specificity and their architectural impact.

Table 5: Vague Requirements Impacting Design

Req ID	Issue	Architectural Impact
REQ-2	Security attributes undefined	Cannot design threat model or security architecture without knowing which attack vectors matter
REQ-6	UX improvements lack metrics	Impossible to validate if architecture achieves goals without quantifiable targets
REQ-7	Scalability criteria unspecified	Cannot determine if design supports required throughput, concurrent users, or data volume
REQ-12	Management improvement undefined	Lacks specific business metrics to measure architecture success
REQ-33	Error handling scope unclear	May result in inconsistent error responses without standardized format
REQ-51	Remote maintenance scope vague	Unclear what hardware control functions are required for implementation

4 Use Case Analysis

4.1 Use Case Coverage Status

The majority of use cases (26 of 28, or 92.9%) have associated requirements and architectural components. Table 6 provides a summary of coverage status.

Status	Count	Percentage
Fully Covered	24	85.7%
Partially Covered	2	7.1%
Not Covered	2	7.1%

Table 6: Use Case Coverage Distribution

4.2 Partially Covered Use Cases

Use Cases Requiring Attention

UC-28 - **Remote Maintenance Management:** Backend task tracking services exist, but the hardware abstraction layer for sending commands to physical devices is absent. The promise of remote control capabilities (unlocking jammed products, diagnostics) cannot be fulfilled without device gateway components.
UC-8 - **Use Case Template Description:** This use case serves as documentation template rather than functional requirement. Contains no actor, flow, or test specifications. Adds no value to traceability.

5 Architectural Quality Assessment

5.1 Layered Architecture Structure

The architecture implements a six-layer pattern with clear separation of concerns. Table 7 details each layer's responsibilities.

Layer	**Responsibility**	**Key Components**
Presentation	UI/UX and user interaction	Mockups, web interfaces, mobile app
Controllers	HTTP routing and input validation	UserController, AdminController, CustomerController
Services	Business logic orchestration	UserService, AdminService, CustomerService
DAO	Data access abstraction	UserDao, TransactionDao, ItemDao, MachineDao
ORM	Object-relational mapping	Hibernate, JPA annotations
Database	Data persistence	PostgreSQL (production), H2 (testing)

Table 7: Six-Layer Architecture

Layering Benefits: Dependencies flow strictly downward; controllers delegate to services; services call DAOs. This structure enables independent layer testing, technology substitution, and clear responsibility boundaries.

5.2 Component Responsibility Issues

Single Responsibility Principle Violations

Services Layer: Described as "contains business logic" without specifying bounded contexts. Risk of accumulating responsibilities across customer purchases, admin configuration, worker maintenance, and user management in a monolithic service layer.
DAO Layer: "Manages data access and retrieval" is too generic. Without specific DAO responsibilities (UserDao handles user CRUD only, TransactionDao handles financial records only), risk of sprawling DAO classes.
Database Component: Overlaps with DAO responsibilities. Distinction between DBManager (connection pooling), DAO classes (query execution), and ORM layer (object mapping) is unclear.

5.3 Design Patterns Application

Pattern application is judicious—Builder only for complex objects, DAOs for all persistent entities, Mappers for domain-database separation. No overengineering observed.

6 Test Strategy Assessment

6.1 Test Infrastructure

The testing stack demonstrates maturity:

Pattern	Application	Benefit
Builder	ConcreteVendingMachine construction	Handles optional parameters; enforces required fields
DAO	UserDao, TransactionDao, ItemDao, MachineDao	Abstracts persistence; enables ORM swapping
Mapper	TaskMapper, ConnectionMapper, InventoryMapper	Separates domain from persistence models

Table 8: Design Patterns

Component	Version	Purpose
JUnit	5.11.0	Unit test execution framework
Mockito	5.18.0	Dependency isolation and mocking
JaCoCo	Latest	Code coverage measurement
H2 Database	In-memory	Fast integration test feedback
PostgreSQL	16	Production database parity

Table 9: Test Infrastructure

6.2 Test Distribution

The test pyramid is properly shaped with majority at unit level for fast feedback, integration tests validating component interaction, and system tests confirming end-to-end flows.

6.3 Test Coverage Strengths

Error Path Testing: Most use cases test multiple failure scenarios (insufficient balance, out of stock, connection errors, not found conditions), ensuring graceful degradation and robust error handling.

DAO Layer Coverage: Each DAO has dedicated CRUD tests verifying persistence operations, with integration tests confirming service-DAO-database interaction.

Service Isolation: Mockito enables service layer testing without database dependencies, providing fast feedback and deterministic test execution.

6.4 Testing Gaps

Table 10: Critical Testing Gaps

Gap Type	Missing Coverage	Risk/Impact
Navigation Tests	No UI flow validation for role-based routing	Navigation bugs reach production
Hardware Integration	No device communication tests	Remote maintenance unverifiable
Performance Tests	No load/stress/concurrency tests	Scalability limits unknown
Security Tests	No SQL injection, auth bypass, or authorization tests	Vulnerabilities undetected
End-to-End Journeys	No multi-use-case sequences	Complete workflows unvalidated

Continued on next page

Table 10 – continued from previous page

Gap Type	Missing Coverage	Risk/Impact
Offline Scenarios	No disconnection/synchronization tests	Offline operation unverifiable

7 PDF Documentation Validation

7.1 Documentation Completeness Assessment

The PDF documentation validation achieved **68.75% feature coverage** against a 0.85 (85%) threshold, indicating incomplete documentation of system features and design decisions.

Metric	Value
Coverage Percentage	68.75%
Threshold Required	85.0%
Coverage Status	Below Threshold
Total Expected Features	16
Features Found	11
Features Missing	5

Table 11: Documentation Validation Results

7.2 Well-Documented Features

Table 12: Well-Documented Features (High Similarity Scores)

Feature	Similarity	Evidence
User-Centric Design	92.7%	Detailed mockups (Fig. 10) showing interactive product catalog with quantity controls
Testing Strategy	91.2%	Comprehensive description of unit and integration testing approach with tool configuration
Standardized Use Cases	91.9%	Documented use case template with ID, name, actors, flows, and test linkage
Defined Pre/Post Conditions	91.9%	Use case specifications include clear state conditions before and after execution
Layered Architecture	90.1%	UML component diagram (Fig. 11) showing six-layer structure with relationships
Role-Based Access Control	88.9%	Clear definition of three user types (admin, worker, customer) with table inheritance
Design Patterns	90.1%	Data Mapper pattern documented for mapper classes separating domain from persistence
Error Handling Process	85.4%	Alternative flows document error scenarios with user-facing messages
Database Design	86.1%	PostgreSQL 16 specified as production database with relational model
Effective Tool Utilization	88.3%	AI tools documented (GitHub Copilot, GPT-4.1, Claude, Gemini) with supervision notes

Continued on next page

Table 12 – continued from previous page

Feature	Similarity	Evidence
Comprehensive Test Coverage	85.8%	Test cases listed for each use case with numbers and verification descriptions

7.3 Undocumented/Poorly-Documented Features

Five expected features have insufficient or missing documentation:

Documentation Gaps

1. **Performance Optimization Strategies:** No documentation of caching, database indexing, or query optimization approaches.
2. **Security Architecture:** Authentication/authorization mechanisms described but encryption, token management, and threat model absent.
3. **API Documentation:** No OpenAPI/Swagger specifications or endpoint documentation provided.
4. **Deployment Architecture:** Infrastructure, containerization, scaling strategies, and monitoring not documented.
5. **Data Migration Strategy:** No documentation of how existing vending machine data integrates or migrates to new system.

Documentation Quality Assessment: While core functional design is well-documented through use cases, mockups, and architecture diagrams, operational and infrastructure aspects lack coverage. This creates implementation ambiguity for deployment, scaling, and security hardening phases.

8 Critical Risks and Recommendations

8.1 Risk Summary

Table 13: Critical Risks Summary

Risk ID	Risk Name	Severity	Affected Items
RISK-1	Offline Operation Unavailability	Critical	REQ-19, REQ-20, REQ-21
RISK-2	Component Responsibility Ambiguity	Critical	Services, DAO, Database layers
RISK-3	Remote Maintenance Unimplementable	Critical	UC-28, REQ-51
RISK-4	Insufficient Documentation	High	Deployment, Security, API specs
RISK-5	Testing Gaps for Edge Cases	High	Navigation, Performance, Security

8.2 Risk 1: Offline Operation Unavailability

Risk: Machines cannot process transactions during network outages, resulting in complete service failure and revenue loss.

Recommendation:

- Design local transaction storage mechanism on vending machine devices
- Define synchronization protocol with conflict resolution
- Architect offline authentication approach (cached credentials or device tokens)
- Implement connection state monitoring and detection

8.3 Risk 2: Component Responsibility Ambiguity

Risk: Vague component definitions lead to inconsistent code placement and architectural degradation.

Recommendation:

- Document precise, single responsibilities for each layer component
- Clarify DAO, service, and database component boundaries
- Consider splitting services layer into bounded domain contexts
- Update architectural diagrams with refined descriptions

8.4 Risk 3: Remote Maintenance Unimplementable

Risk: Backend task tracking exists but hardware abstraction layer for device control is absent.

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

8.5 Risk 4: Insufficient Documentation

Risk: Missing documentation for deployment, security, and API specifications creates implementation ambiguity.

Recommendation:

- Create OpenAPI/Swagger specifications for all endpoints
- Document security architecture (authentication, encryption, threat model)
- Specify deployment architecture (containerization, scaling, monitoring)
- Define data migration strategy for existing systems

8.6 Risk 5: Testing Gaps for Edge Cases

Risk: Navigation flows, performance limits, and security vulnerabilities remain unvalidated.

Recommendation:

- Add navigation integration tests for role-based routing
- Implement end-to-end tests spanning multiple use cases
- Add performance testing for scalability baseline
- Introduce security tests for common vulnerabilities

9 Conclusion

9.1 Overall Assessment

The JavaBrew architectural blueprint demonstrates strong fundamentals: comprehensive layered architecture, extensive test coverage (158 tests), effective design patterns, and complete traceability from requirements through tests. The 92.9% use case coverage and 74.5% requirements coverage indicate a mature development process.

However, three critical gaps threaten production viability:

1. **Offline Operation:** Three requirements (REQ-19, REQ-20, REQ-21) for network outage handling are architecturally unsupported, creating complete service failure during disconnections.
2. **Component Clarity:** Vague responsibility definitions in services, DAO, and database layers risk architectural degradation as complexity increases.
3. **Remote Maintenance:** Backend infrastructure exists but hardware abstraction layer for device control is absent, making promised remote capabilities undeliverable (UC-28).

Documentation validation revealed 68.75% feature coverage (below 85% threshold), with gaps in deployment architecture, security specifications, and API documentation.

9.2 Recommended Actions

Table 14: Prioritized Action Items

Priority	Action Item	Timeline	Related Item
P0	Design offline operation architecture	2-3 weeks	RISK-1
P0	Refine component responsibilities	1 week	RISK-2
P0	Define remote maintenance scope	2 weeks	RISK-3
P1	Create API/OpenAPI documentation	1-2 weeks	RISK-4
P1	Document security architecture	1 week	RISK-4
P1	Add navigation integration tests	1 week	RISK-5
P2	Clarify vague requirements	1 week	Section 3.2
P2	Add performance testing	2 weeks	RISK-5
P3	Define deployment architecture	1-2 weeks	RISK-4
P3	Add security vulnerability tests	1 week	RISK-5

9.3 Final Assessment

This architecture provides a solid foundation suitable for controlled deployment with reliable network connectivity. Addressing offline operation support, clarifying component boundaries, and completing documentation will elevate the design to production-grade robustness for diverse deployment scenarios. The automated traceability analysis proves valuable for identifying these gaps early, before implementation costs make corrections expensive. Maintaining traceability discipline as the system evolves will continue to provide quality assurance benefits throughout the software lifecycle.