# Software Architecture and Design Specification

**Project:** Mini Flipkart (E-commerce Platform)

**Version:** 2.0

**Authors:** Anand MS, Nahush P Shetty, Machiraju Karthikeya, Gururaja Rao M, Krishna Venkatesh

**Date:** 2025-11-23

**Status:** Draft

## Revision History

| **Version** | **Date** | **Author** | **Change Summary** |
| --- | --- | --- | --- |
| 1.0 | 2025-11-23 | Team | Initial document creation based on implemented system |

## Approvals

| **Role** | **Name** | **Signature/Date** |
| --- | --- | --- |
| Course Coordinator | Prof Rajesh Banginwar |  |
| QA Lead | Nahush P Shetty |  |
| Product Owner | Gururaja Rao M |  |

# 1. Introduction

## 1.1 Purpose

This document specifies the architecture and design of the Mini Flipkart e-commerce platform. It provides a comprehensive overview of the system's structure, components, design patterns, and implementation details.

## 1.2 Scope

This system covers the following major functional areas:

* **Customer Features**: User registration, authentication, product browsing, advanced filtering, product detail pages, shopping cart, checkout flow, payment processing, order tracking, and personalized recommendations.
* **Order Management**: Order orchestration, inventory synchronization, shipment tracking, and compensation handling.
* **Security & Compliance**: Audit logging, tamper detection, fraud detection, security dashboard, and compliance monitoring.
* **Administrative**: Inventory dashboard, security dashboard, and system monitoring.

## 1.3 Audience

The primary audience includes Software Engineers, Quality Assurance Engineers, System Architects, Product Managers, Business Analysts, and Academic Evaluators.

## 1.4 Definitions

* **SKU**: Stock Keeping Unit - unique identifier for inventory items.
* **API**: Application Programming Interface.
* **JWT**: JSON Web Token - authentication mechanism.
* **PII**: Personally Identifiable Information.
* **RBAC**: Role-Based Access Control.
* **PCI-DSS**: Payment Card Industry Data Security Standard.
* **GDPR**: General Data Protection Regulation.
* **CI/CD**: Continuous Integration/Continuous Deployment.

# 2. Document Overview

## 2.1 How to Use This Document

This document provides key deliverables, including:

* **Architectural Deliverables**: UML diagrams, component descriptions, architectural patterns.
* **Design Specifications**: API contracts, data models, error handling strategies.
* **Security Architecture**: Threat models, security controls, compliance measures.
* **Traceability**: Mapping of requirements to implementation (Jira task IDs: MF-13 through MF-72).

## 2.2 Related Documents

* Software Requirements Specification (SRS)
* Software Test Plan (STP)
* Requirements Traceability Matrix (RTM)
* Jira Backlog: docs/jira-backlog.md
* Logging Policy: LOGGING\_POLICY.md - Audit and security logging standards.

# 3. Architecture

## 3.1 Goals & Constraints

### Goals

* **Availability**: System shall maintain 99.9% uptime during business hours.
* **Performance**: 95% of API requests shall respond within 3 seconds.
* **Scalability**: Support 10,000 concurrent users with horizontal scaling capability.
* **Security**: PCI-DSS compliant payment processing and GDPR-compliant data handling.
* **Maintainability**: Modular architecture enabling parallel development by 4+ engineers.

### Constraints

* **Technology Stack**: Node.js/Express.js backend, React.js frontend, MongoDB database.
* **Compliance**: Must adhere to PCI-DSS and GDPR.
* **Platform**: Mobile-responsive web application only (no native mobile apps in v1.0).
* **Time**: Two-sprint delivery cycle.

## 3.2 Stakeholders & Concerns

| **Stakeholder** | **Primary Concerns** |
| --- | --- |
| Customers | Security of personal data, system availability, fast checkout. |
| Administrators | System monitoring, platform integrity, compliance reporting. |
| Developers | Modularity for parallel development, maintainability, clear API contracts. |
| QA Engineers | Testability, clear acceptance criteria, automated test coverage. |

## 3.3 Chosen Architecture Pattern and Rationale

**Chosen Pattern**: **Three-Tier (Layered) Architecture**

**Rationale**:

1. **Clear Separation of Concerns**: Distinct Presentation (React), Business Logic (Express/Services), and Data Access (MongoDB/Mongoose) layers.
2. **Modularity**: Enables parallel development and independent testing.
3. **Scalability**: Each tier can be scaled horizontally and independently.

## 3.4 Technology Stack & Data Stores

| **Category** | **Technology** | **Key Components** |
| --- | --- | --- |
| **Frontend** | React.js 19.2.0, Redux 5.0.1, Axios 1.7.9 | State Management, Routing, UI, HTTP Client |
| **Backend** | Node.js 20.18.0, Express.js 4.19.2 | Runtime, Web Framework |
| **Database** | MongoDB 8.6.1, Mongoose 8.6.1 | Primary Data Store, ODM |
| **Auth/Security** | JWT, bcryptjs, express-validator | Authentication, Hashing, Validation |

## 3.5 Component Descriptions

### 3.5.1 Frontend Web Application

* **Technology**: React.js, Redux.
* **Key Components**: Authentication, Product Catalog, Shopping Cart, Checkout, Orders, Tracking, Recommendations, and Admin Dashboards.

### 3.5.2 Authentication Service

* **Functionality**: User registration, **JWT-based authentication**, password hashing with bcryptjs.
* **Endpoints**: POST /api/auth/register, POST /api/auth/login.

### 3.5.3 Order Processing Service

* **Functionality**: Order submission with **orchestration**, idempotency handling, payment processing integration, and **compensation handling** for failures (MF-47).

### 3.5.4 Audit Logging & Compliance Service

* **Functionality**: **Immutable audit logs**, **tamper detection** (MF-71), and **PII redaction** (MF-67).

# 4. Design

## 4.1 API Design Summary

| **Area** | **Endpoint** | **Description** | **Key Features** |
| --- | --- | --- | --- |
| **Auth** | POST /api/auth/login | Authenticate user. | Returns JWT token. |
| **Products** | GET /api/products | List all products. | Supports advanced **filtering** (category, price range, sort). |
| **Orders** | POST /api/orders/submit | Submit a new order. | Requires Authorization and optional **Idempotency-Key** (MF-43). |
| **Tracking** | GET /api/orders/:id/tracking | Get shipment status. | Returns tracking timeline and estimatedDelivery. |

## 4.2 Error Handling, Logging & Monitoring

### Error Handling Strategy

* **Standardized Error Response**: { "success": false, "error": "...", "errors": [...] }.
* **HTTP Status Codes**: Uses 400 Bad Request, 401 Unauthorized, 404 Not Found, and 409 Conflict (for idempotency).

### Logging Strategy

* **Log Retention**: Security logs: **7 years**. Audit logs: 5 years.
* **PII Redaction** (MF-67): Email addresses and phone numbers are redacted in logs. Passwords and payment data are never logged.
* **Audit Logging**: **Immutable audit logs** with checksums (MF-70) for critical events.

# 5. Appendices

## 5.1 Security Architecture

### Threat Modeling (STRIDE)

| **Threat** | **Mitigation** | **Implementation** |
| --- | --- | --- |
| **Spoofing** | Strong authentication | JWT tokens (MF-15), bcrypt password hashing (MF-17). |
| **Tampering** | Input validation | express-validator (MF-17), tamper detection service (MF-71). |
| **Repudiation** | Audit logging | Immutable audit logs (MF-69), checksum verification (MF-71). |
| **Information Disclosure** | Encryption | **TLS** for data in transit, PII redaction (MF-67). |
| **Elevation of Privilege** | RBAC | Role-based access control (user/admin roles). |

## 5.2 Development Workflow

### Git Workflow

* **Branch Strategy**: Each Jira subtask (MF-13 through MF-72) gets its own branch (MF-<task-id>).
* **Merge Strategy**: Subtasks $\rightarrow$ Story branch $\rightarrow$ Epic branch $\rightarrow$ develop $\rightarrow$ main.

### Testing Strategy

* **Unit Tests**: Focused on the service layer (MF-18, MF-24, etc.).
* **Integration Tests**: For API endpoints (Supertest) and end-to-end user flows (MF-42).

**Document End**