Software Architecture and Design Specification

Project: E-Commerce Software(SnapBuy)

Version: 1.0

Authors: <Neha PM (PES1UG23AM183)> <Raksha (PES1UG23AM229)> <Rakshitha M (PES1UG23AM231)>

Date: 21-09-2025

Status: Draft

Revision History

Version	Date	Author	Change Summary
1.0	21 Sep 2025	Team API	Initial Draft

Approvals

Role	Name	Signature/Date
Architecture Lead	Neha PM	neha – 21.09.20205
Development Lead	Rakshitha M	rakshitha – 21.09.2025
Product Owner	Prof. Arpitha K	

1. Introduction

1.1 Purpose

This document describes the architecture and design of the SnapBuy E-Commerce Platform. It provides a comprehensive overview of the system's structure, components, technologies, and the rationale behind key design decisions to guide development, testing, and maintenance.

1.2 Scope
This document covers the architecture for all in-scope features from the SRS, including user authentication, product catalog, shopping cart, checkout, order management, notifications, and the admin dashboard. It excludes out-of-scope items like third-party logistics and warehouse automation.

1.3 Audience
Developers, Testers, Evaluators, and future maintenance teams.

1.4 Definitions
MVC(Model-View-Controller), REST(Representational State Transfer), API(Application Programming Interface), PCI-DSS(Payment Card Industry Data Security Standard), HTTPS(Hypertext Transfer Protocol Secure)

2. Document Overview

2.1 How to use this document
This document serves as the single source of truth for the system's architecture. It should be used by developers to understand component interaction, by testers to plan integration tests, and by reviewers to evaluate the system's design quality and security.

2.2 Related Documents
SnapBuy Software Requirements Specification (SRS), Software Test Plan (STP), Requirements Traceability Matrix (RTM)

3. Architecture

3.1 Goals & Constraints

Goals: Secure, scalable, maintainable, and provide a good user experience.

Constraints: Limited development time, must comply with data privacy (GDPR-inspired) and payment security (PCI-DSS) standards, must integrate with third-party payment gateways.

3.2 Stakeholders & Concerns

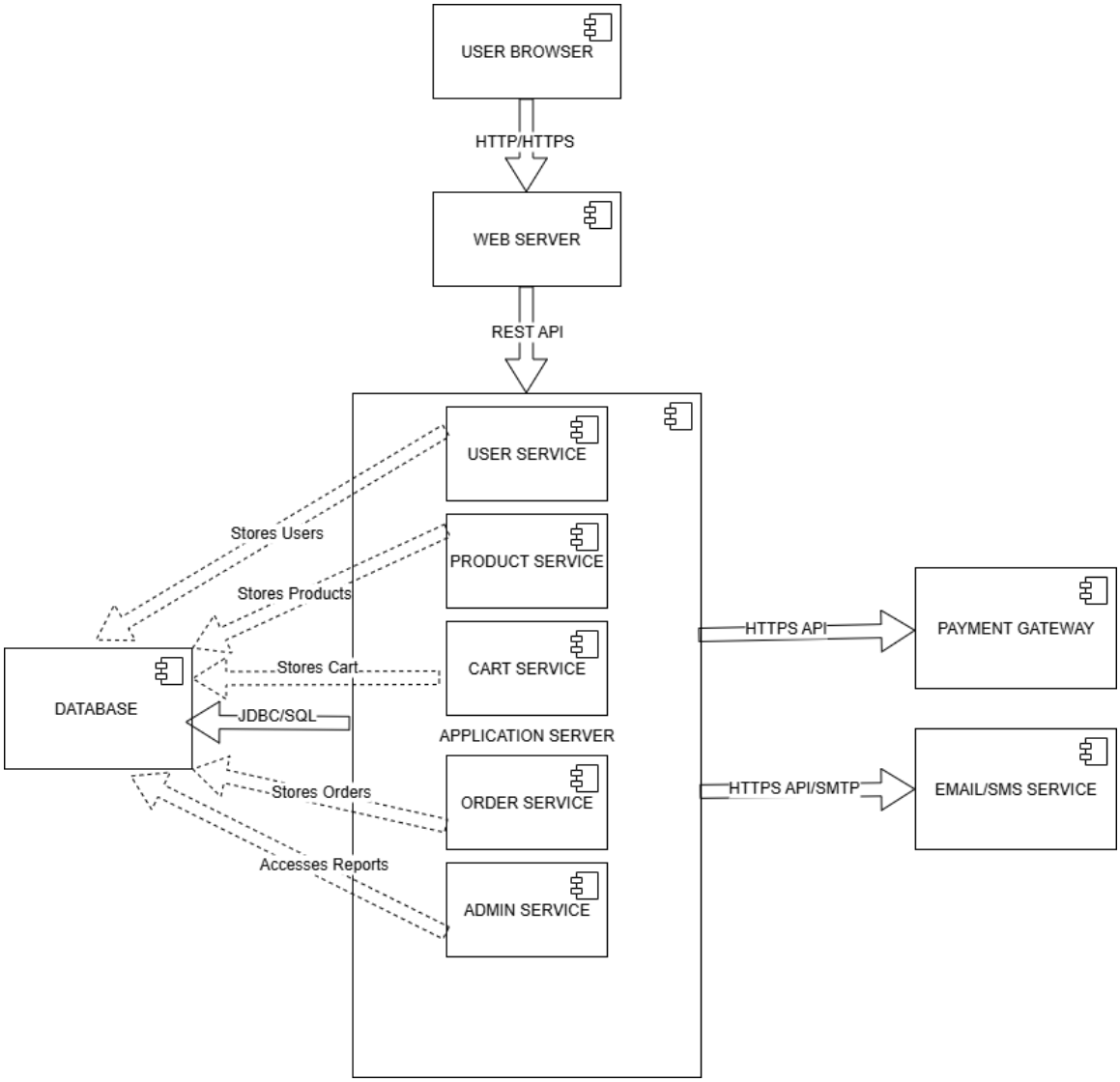
Customers: Security of personal/financial data, performance, ease of use.

Administrators: Reliability, accuracy of reports, ease of inventory management.

Developers: Modularity, clear APIs, ease of debugging and feature addition.

Evaluators: Adherence to SRS, architectural best practices.

3.3 Component (UML) Diagram



3.4 Component Descriptions

- Web Server: Serves static files and acts as a reverse proxy.
- Application Server: Contains controllers for handling HTTP requests and services for:
  - User Service: Handles registration, authentication, and profiles.
  - Product Catalog Service: Manages product data, search, and categories.
  - Shopping Cart Service: Manages cart operations and calculations.
  - Order Service: Processes orders, payments, and manages order status.
  - Notification Service: Sends emails and SMS.
  - Admin Service: Handles backend operations for inventory and reports.
- Database: Stores all persistent data.
- External Services: Payment Gateway API, Email/SMS API.

3.5 Chosen Architecture Pattern and Rationale

Architecture: Layered architecture chosen.

Rationale: Well-understood, provides clear separation of concerns, easier to develop and test for a small team within a limited timeframe. Micro Services were rejected due to their inherent complexity in deployment, monitoring, and inter-service communication, which is unnecessary for a project of this scale.

3.6 Technology Stack & Data Stores

Frontend: HTML5, CSS3, JavaScript

Backend: Node.js with Express.js

Database: MySQL

APIs: RESTful APIs for internal communication, JSON for data exchange.

Security: TLS 1.2+ (HTTPS), bcrypt for password hashing.

Hosting: AWS EC2(for application), AWS RDS(for database).

3.7 Risks & Mitigations

- Risk: Single point of failure at the database layer.

  Mitigation: Implement regular database backups. For a more advanced mitigation, consider database replication in the future.

- Risk: Payment gateway API downtime during checkout.

  Mitigation: Implement graceful error handling and user-friendly messages. Use a sandbox for development and testing.

3.8 Traceability to Requirements

FR-001 (User Login) → User Service, Authentication Controller

FR-004 (Add to Cart) → Cart Service, Cart Controller

FR-007 (Payment Modes) → Order Service, Payment Gateway API integration

FR-012 (Admin Manage Products) → Admin Service, Product Controller

NFR-001 (Search Performance) → Product Catalog Service, Database indexing

SR-001 (Password Hashing) → User Service

3.9 Security Architecture

Threat Modeling (STRIDE):

Spoofing → Prevented by secure password hashing and session management.

Tampering → Prevented by using HTTPS (TLS) on all communications.

Info disclosure → Prevented by hashing passwords and not logging sensitive data.

Denial of Service → Mitigated by input validation and rate limiting on login/checkout.

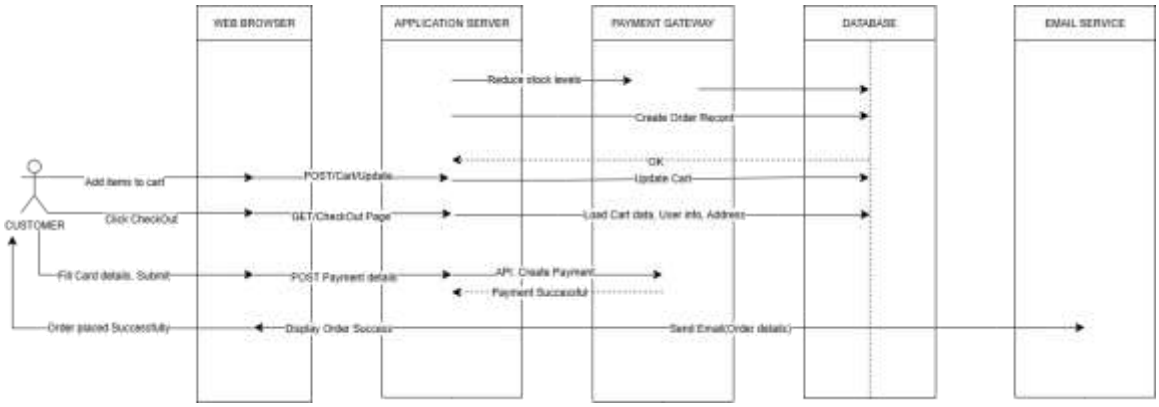
Elevation of Privilege → Prevented by strict role-based access control.

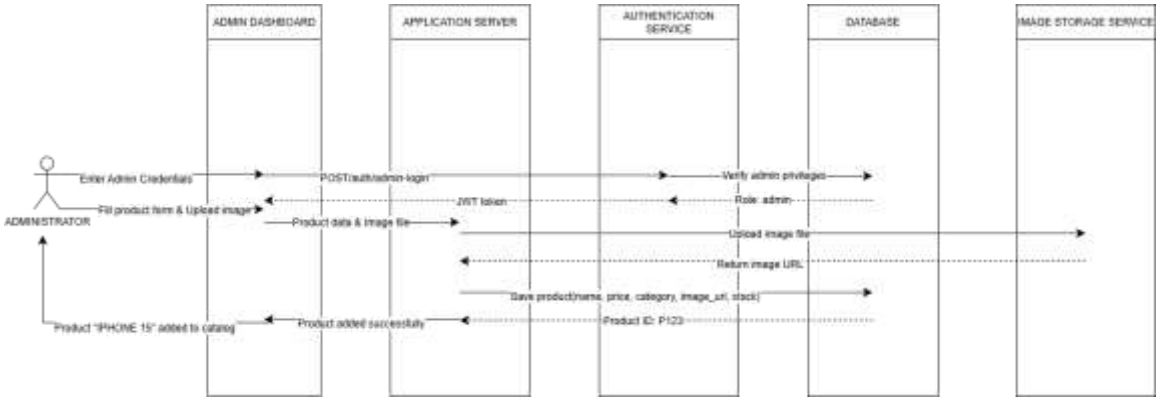
4. Design

4.1 Design Overview

The system follows an MVC design within its layered architecture for modularity. The frontend views are built to be responsive, and the backend controllers handle request routing, delegating business logic to service classes.

4.2 UML Sequence Diagrams





4.3 API Design

- Endpoint: /api/auth/login

Method: POST
Request: { "email": "user@example.com", "password": "plaintext_password" }
Response (200): { "token": "jwt_token", "userId": 123 }
Errors: 401 Unauthorized (Invalid credentials)

- Endpoint: /api/orders
  Method: POST
  Headers: Authorization: Bearer <jwt_token>
  Request: { "items": [{"productId": 45, "qty": 2}], "shippingAddress": {...}, "paymentMethod": "card" }
  Response (201): { "orderId": 1001, "status": "confirmed", "total": 2999 }
  Errors: 400 Bad Request (Cart empty), 402 Payment Required (Payment failed)

4.4 Error Handling, Logging & Monitoring

- Error Handling: Consistent JSON error responses
- Logging: Log all errors and critical events. Never log passwords, card numbers or PINs.
- Monitoring key metrics: API response times, error rates, and system uptime.

4.5 UX Design

The UI will be designed to be intuitive and responsive, ensuring a consistent experience on both desktop and mobile devices. Key user flows will be streamlined to minimize steps.

4.6 Open Issues & Next Steps

Open Issue: Final decision on payment gateway vendor.

Next Steps:

- Finalize technology stack.
- Set up development environment and CI/CD pipeline.
- Begin sprint planning for core modules.

5. Appendices

5.1 Glossary: MVC, REST, JWT, Bcrypt.

5.2 References: REST API Tutorial, OWASP Top 10 Web Application Security Risks, PCI Security Standards Council.

5.3 Tools: Visual Studio Code, draw.io, Swagger.