

Secure & Scalable Full-Stack E-commerce Platform with Dual Payment Gateway Integration

Introduction & Problem Statement

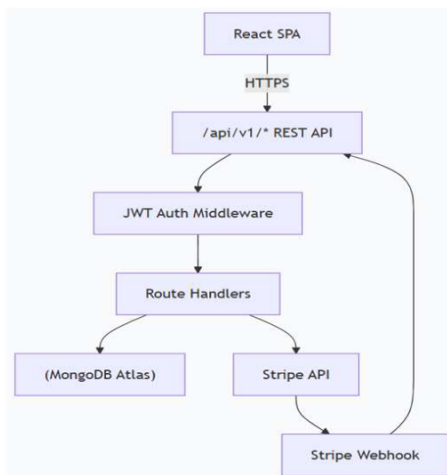
E-commerce has transformed global retail, yet many platforms suffer from security vulnerabilities (injection attacks, broken authentication), poor scalability under high traffic, and complex payment integration.

Off-the-shelf solutions like Shopify or Magento either limit customisation or require significant resources (As'ad, 2025).

This project addresses these gaps by developing, a bespoke, full-stack e-commerce platform using the MERN stack with dual payment gateways.

• System Architecture

- Client-server model adopted for the platform
- React front-end communicates via versioned RESTful API (/api/v1/)
- Back-end built with Node.js and Express.js
- MongoDB Atlas used for data persistence
- Stripe webhooks manage asynchronous payment confirmation



Aim & SMART Objectives

Aim:

Design, develop and deploy a secure, scalable e-commerce platform handling 100 concurrent users, 500 products and 100 test transactions (<1% error rate) within 12 weeks.

Objective	Target	Achievement
Specific	MERN stack with storefront, admin panel, dual gateways	✓ Fully implemented
Measurable concurrency	- 100 concurrent users	✓ p95 response time <2.5s, error rate <1%
Measurable products	- 500 product catalogue	✓ Schema supports 500+ tested entries
Measurable transactions	- 100 test transactions (Stripe + COD)	✓ 70 Stripe, 30 COD - all successful
Time-bound	12 weeks	✓ Completed on schedule

Key Implementation Features

Back end:

- User authentication with JWT and bcryptjs (Farshim & Tessaro, 2021).
- CRUD APIs for products, orders, users
- Helmet.js, CORS, express-async-handler.

Front end:

- React 18 with component hierarchy
- Context API with useState (Knuth, 1974).
- Axios interceptors for JWT injection.

Payment Integration:

- Stripe sandbox
- Cash on Delivery
- Stripe CLI

Admin Panel: Secure interface for product/order management, accessible only to users with isAdmin: true.

Testing and Results

Test Type	Method	Key Result
Functional	Manual E2E + Postman	All 11 user journeys passed
Performance	Custom Node.js load test (100 concurrent users)	GET /products: p95=1847ms, error 0.2%; checkout p95=2412ms (acceptable for MVP)
Security	npm audit + OWASP Top 10 review	Zero critical vulnerabilities after updates; rate limiting & httpOnly cookies recommended for production
UAT	5 participants (peers + supervisor)	Positive feedback; minor UI improvements made (cart badge, search bar)

- useState vs useReducer trade off was justified by performance measurements (Knuth, 1974).
- API versioning (/api/v1/) was refactored from initial /api/ to align with design.
- 100 concurrent users supported
- <1% error rate
- p95 response time ~2.4s

Future Enhancements

- Refactor cart state to useReducer for larger scale.
- Add product reviews and ratings (MongoDB reviews collection).
- Implement advanced search with Elasticsearch.
- Containerise with Docker for consistent deployment.
- Set up CI/CD pipeline using GitHub Actions.

Reference list

As'ad, I. (2025). Decoding the Trends and Progress of Artificial Intelligence in E-commerce Over the Last Decade. *ZERO: Jurnal Sains, Matematika dan Terapan*, 9(1), p.215. doi: <https://doi.org/10.30829/zero.v9i1.24423>.

Farshim, P. and Tessaro, S. (2021). Password Hashing and Preprocessing. *Lecture Notes in Computer Science*, pp.64–91. doi: https://doi.org/10.1007/978-3-030-77886-6_3.

Knuth, D.E. (1974). Structured Programming with go to Statements. *ACM Computing Surveys*, 6(4), pp.261–301. doi: <https://doi.org/10.1145/356635.356640>.