WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

"KREEDA KALPA SPORTKART: a smart E-commerce website for sport accessories"

Shashidhara M S *[1],Praneeth Prasad *[2], Lokanatha Swamy T*[3], Manoj K G *[4]

*[1]Associate Professor, Department of Computer Science and Engineering, KIT, Tiptur, Karnataka, India
*[2]Student, Department of Computer Science and Engineering, KIT, Tiptur, Karnataka, India
*[3]Student, Department of Computer Science and Engineering, KIT, Tiptur, Karnataka, India
*[4]Student, Department of Computer Science and Engineering, KIT, Tiptur, Karnataka, India

ABSTRACT :

Buyers of sports equipment in small towns must walk from shop to shop to compare specifications, prices and availability, while independent shop owners rarely have an online presence. This paper presents SportKart, a web-based hyperlocal sports e-commerce platform that connects customers with nearby sports shops. Customers detect or choose a location, find shops within a chosen radius using the Haversine formula, search, filter and compare products, place orders that are split by shop, and follow order status in real time. Sellers register a shop, manage products and stock, receive new-order alerts and view dashboard metrics. The system uses a Next.js and React front end with Redux Toolkit and Leaflet maps, and a Node.js and Express back end with Socket.io, JWT authentication, bcrypt password hashing and role-based access control. Recommendations come from a transparent rule-based scoring function. Data is held in an in-memory store seeded with ten shops and thirty products. Of 15 unit tests and 59 API tests, 73 passed; the single failure exposed a missing role check on the order-status route. Read requests averaged 1.7 to 3.2 ms on a single-core machine. Payment processing, persistent storage and user evaluation remain future work.

Keywords: Hyperlocal E-Commerce, Haversine Distance, Next.js, Node.js, Socket.io, Role-Based Access Control, Recommendation Scoring.

INTRODUCTION

Sports equipment is a category in which buyers care about specific physical attributes: the willow grade of a cricket bat, the string tension a badminton racket is rated for, or the cushioning of a running shoe. In a small town such as Tiptur in Karnataka, a customer who wants to compare these attributes normally walks from one sports shop to another, asks about availability at each counter and remembers the prices quoted. Shop owners, in turn, rarely have a digital presence through which nearby customers can see what is in stock, and they have little visibility of their own sales and stock position.
Studies of local online platforms report that many of them struggle to attract enough customers, and that usability, communication features and a clear benefit for shop owners decide their success [1]. Empirical work on hyperlocal marketplaces further shows that buyer engagement and seller trust signals strongly influence whether a transaction succeeds [2]. These findings motivate a platform that serves both sides of a local market.
This paper presents **SportKart**, a web-based hyperlocal sports e-commerce platform. Here, hyperlocal means that the system restricts its attention to a small area around the customer: shops are discovered by distance, products are compared across the shops of that neighbourhood, and an order is placed with the shop that physically holds the stock. The paper makes the following contributions:

1. A distance-aware shop discovery module that applies the Haversine formula to every shop and returns those within a chosen radius, nearest first, together with a map view.
2. An order pipeline that groups a multi-shop cart into one order per shop, computes fees and discounts on the server, and deducts stock.
3. A Socket.io room design that pushes new-order alerts to sellers and status updates to customers without page reloads.
4. A transparent rule-based recommendation function suited to a platform that has too little interaction data for collaborative filtering.
5. A candid, test-based account of what works and what does not: 73 of 74 tests passed, and the defects found are documented with suggested corrections.

The contribution is the integrated design and its honest evaluation, not a new algorithm. Section II reviews related work; Section III states the problem and objectives; Sections IV and V describe the methodology and the system design; Section VI reports the results; Section VII lists the limitations; and Section VIII concludes the paper.

LITERATURE REVIEW

A. Local and hyperlocal commerce

Schade et al. analysed local online platforms in Europe that connect brick-and-mortar retailers with local customers and found that many have low customer numbers; success factors included multi-way communication and a stable financial model [1]. Hong et al. studied a very large set of transactions on a hyperlocal marketplace and showed that buyer engagement and seller trust signals strongly influence transaction success [2]. SportKart therefore displays shop ratings, a verified indicator, opening status and product reviews, and gives sellers an operational benefit through a

stock console and order alerts.

B. Recommender systems

Resnick and Varian introduced recommender systems as tools that use the opinions or behaviour of a community to help individuals find items of interest [3]. Schafer et al. described how e-commerce sites apply them as ranked lists, related-item suggestions and personalised pages [4]. Item-based collaborative filtering computes item-to-item similarity from rating data and scales to large catalogues [5], and was popularised at a large online retailer [6]. Recommenders are commonly classed as content-based, collaborative or hybrid, and all suffer from cold-start and sparse-data problems [7], [8]. Because a new local platform has few users, SportKart uses content- and rule-based scoring instead of collaborative filtering.

C. Enabling technologies

The Haversine formula gives the great-circle distance between two points on a sphere from their latitudes and longitudes [9]. Passwords are protected with the adaptive bcrypt scheme [10], and signed JSON Web Tokens carry user claims between client and server [11]. The WebSocket protocol provides a persistent two-way connection [12], and Socket.io adds named rooms and events on top of it [13]. The REST architectural style, based on resources and stateless interaction, organises the API [14].

Table 1: Related work and the gap addressed by SportKart.

Work	Focus	Relevant limitation
Schade et al. [1]	Local online platforms in Europe	Survey; no implemented system; many platforms lack customers
Hong et al. [2]	Hyperlocal marketplace transactions	Analyses behaviour; does not provide a shop or seller toolset
Resnick, Schafer [3], [4]	Recommender systems in e-commerce	General concepts; need interaction data
Sarwar, Linden [5], [6]	Item-based collaborative filtering	Needs rating data; cold start on a new platform
Adomavicius, Burke [7], [8]	Content-based and hybrid recommenders	Survey level; no local-shop context
This paper	Local discovery, split orders, real-time tracking, rules	Prototype; in-memory data; no payment processing

Three observations follow. First, the surveyed work treats local commerce, recommendation and real-time communication separately; a small-town sports platform needs all three together. Second, collaborative filtering cannot be applied directly to a new platform [7], [8]. Third, event channels suit order tracking better than repeated polling [12].

PROBLEM STATEMENT AND OBJECTIVES

A. Problem statement

To design and develop a web-based hyperlocal sports e-commerce platform that lets customers in a small town discover nearby sports shops by distance, search, filter and compare products across those shops, place orders with the selected shops and follow order progress in real time, and that lets shop owners manage their catalogue,

stock and orders and view basic business metrics, using token-based authentication and role-based access control.

B. Objectives

1. Discover shops near a chosen or detected location, filtered by radius, opening status and rating, and show them on an interactive map.
2. Provide a searchable, filterable catalogue across twelve sports categories, with side-by-side comparison of up to four products and price comparison across nearby shops.
3. Support a complete purchase flow: cart, wishlist, three-step checkout and one order per shop.
4. Give real-time feedback through Socket.io events, an order timeline and notifications.
5. Give sellers a console for catalogue management, quick stock updates with low-stock alerts, order handling and a dashboard.
6. Provide rule-based personalised and related-product recommendations.
7. Secure the API with hashed passwords, JWT authentication and role checks, using only open-source technologies that run on a student laptop.

PROPOSED METHODOLOGY

A. Development approach

SportKart was built incrementally as vertical slices. Each module (authentication, shops, products, orders, seller console, recommendations) was added together with its route, controller and pages. The system consists of two processes, a Node.js back end on port 5000 and a Next.js front end on port 3000. Seed data provides demonstration customer and seller accounts so every feature can be exercised without a database. Figure 1 shows the five-step customer journey that the platform supports.

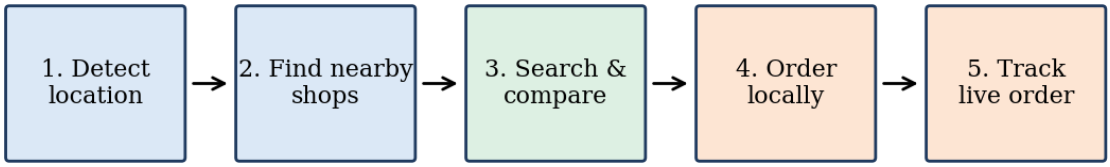


Figure 1: SportKart customer journey.

B. Distance calculation and nearby-shop search

The distance between the customer and a shop is the great-circle distance from the Haversine formula [9]. With latitudes φ and longitudes λ in radians and Earth radius R = 6371 km:

$$a = \sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2) \quad (1)$$

$$d = 2R \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)}) \quad (2)$$

The result is rounded to 0.1 km. The nearby search applies this function to every shop, keeps shops with d not greater than the radius and sorts them in ascending order of distance. Optional filters keep only open shops or shops above a minimum rating. The default location is the centre of Tiptur (13.2575° N, 76.4782° E).

C. Recommendation scoring

The recommender is content-based and rule-based; no model is trained. In product-page mode, each candidate is scored by its relation to the current product. Otherwise the score is the product rating times ten (4.5 if missing) plus boosts from the user profile. Table 2 lists the weights as coded. Each result carries a short reason string, such as “Based on your interest in Cricket”.

Table 2: Scoring rules of the recommendation service.

Mode	Signal	Score added
Product page	Same category / complementary category	+60 / +50

Mode	Signal	Score added
Product page	Same brand; rating ≥ 4.7; featured	+25; +15; +10
Profile / guest	Base score	rating × 10 (4.5 if missing)
Profile	Browsing-history item of a category	+15 per item
Profile	Preferred category	+25
Profile	Search keyword matches a category	+20 per matching query
Profile / guest	Featured; discount ≥ 15 percent	+15; +10

D. Order pricing and splitting

On order creation the server looks up each cart item, uses its own price and shop whenever the product is found, groups the items by shop and creates one order per group. For an order whose item subtotal is S, the delivery fee F is zero if S exceeds 999 rupees and 40 rupees otherwise; the discount D is 150 rupees if S exceeds 2000 and zero otherwise; and the grand total is

$$G = S + F - D \quad (3)$$

The data store then deducts the ordered quantity from the stock of each product with enough stock. Order status moves through Order Placed, Seller Accepted, Preparing, Packed and Out for Delivery to Delivered; Cancelled is the other end state. Every change is appended to a status timeline.

E. Evaluation method

The delivered project contained no automated tests, so two test groups were written: 15 unit tests of pure functions and the data store, and 59 integration tests that send HTTP requests to a running server and, in one case, open a Socket.io connection. MongoDB was not running, so the server used its in-memory store. Response times were measured with 200 sequential requests per endpoint (50 for login) after five warm-up requests, from a client on the same single-core machine with about 4 GB of memory. Black-box and white-box techniques were followed [15].

SYSTEM DESIGN AND IMPLEMENTATION

A. Architecture

SportKart follows a three-tier client-server architecture (Figure 2). The client is a Next.js 14 application [16] with 28 routes written in React 18. A Redux Toolkit store holds six slices (authentication, cart, wishlist, comparison, location and toast), and the first four persist to localStorage. Maps use react-leaflet [17] with OpenStreetMap tiles [18]. The server is a Node.js process running Express 5, with a Socket.io server attached to the same HTTP server so that REST and WebSocket traffic share port 5000. Protected routes pass through an authenticate middleware, which verifies the JWT, and a requireRole middleware, which compares the role claim with the roles allowed for the route.

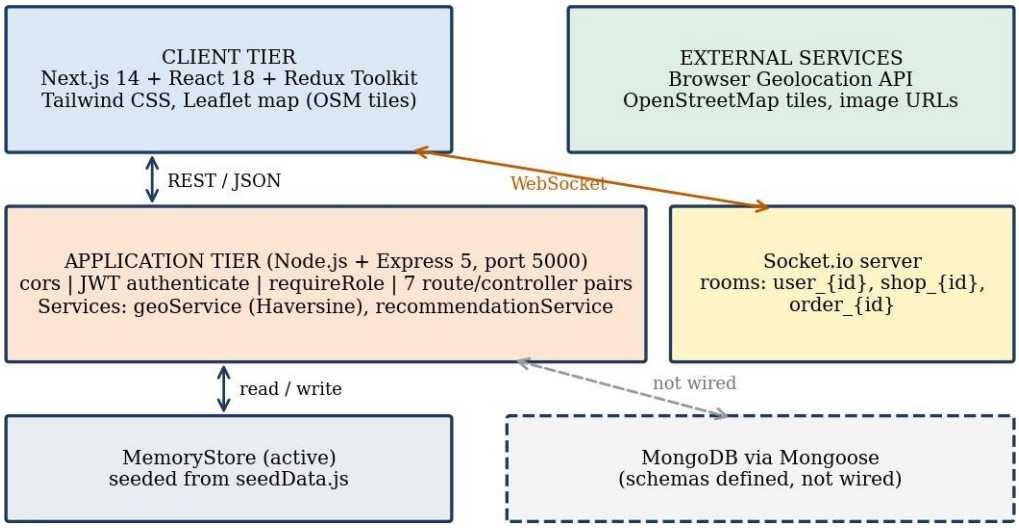


Figure 2: System architecture of SportKart as implemented.

Table 3: Technology stack (versions from package.json).

Layer	Technology	Version	Purpose
Front end	Next.js, React	14.2.15, 18.3.1	App Router pages and components
Front end	Redux Toolkit	2.2.8	Auth, cart, wishlist, comparison, location, toast
Front end	Tailwind CSS	3.4.14	Responsive utility-first styling
Front end	Leaflet, react-leaflet	1.9.4, 4.2.1	Map, markers and radius circle
Back end	Express, Socket.io	5.2.1, 4.8.1	REST API, rooms and events
Back end	jsonwebtoken, bcryptjs	9.0.3, 3.0.3	JWT (7-day expiry), hashing (10 rounds)
Data	MemoryStore; Mongoose	custom; 9.3.0	Active store; schemas, not wired

B. Real-time design

After login, the client emits join_user with the user id and, for sellers, join_shop with the shop id; the tracking page emits join_order. Controllers then emit order:new to the shop room, notification:new to the user room and order:statusUpdated to the order room. A low-stock update raises a LOW_STOCK notification to the shop room.

C. Data model

Six Mongoose schemas (User, Shop, Product, Order, Review, Notification) describe the intended persistent structure. A shop location is a GeoJSON point with a 2dsphere index, and order items and addresses are embedded sub-documents. In the delivered system, however, every controller reads and writes a MemoryStore object seeded from seedData.js, so all data created at run time is lost when the server restarts. This is the most important architectural limitation of the prototype.

D. Implementation status

Every feature was checked against the source code. Table 4 records whether it is fully implemented, partially implemented or absent.

Table 4: Implementation status of features.

Feature	Status	Evidence and remarks
Registration, login, JWT, bcrypt	Implemented	Tests T02–T11; weaknesses in Table 5
Role-based access control	Implemented	Product, shop and seller routes; order-status route unrestricted (T49)
Catalogue, filters, search, compare	Implemented	Tests T12–T21, U10–U13
Nearby shops (Haversine)	Implemented	Tests T22–T25, U02–U06
Map, GPS, cart, wishlist	Implemented	Code inspected; not run
Orders split by shop, stock deduction	Implemented	Tests T38–T44, U14
Real-time alerts and tracking	Implemented	Test T45 observed order:statusUpdated
Reviews, rule-based recommendations	Implemented	Tests T50–T57; rating not recalculated
Seller console, low-stock alert	Implemented	Tests T26–T37; low-stock code inspected

Seller dashboard and analytics	Partial	Fixed weekly series and fallback totals
Payment processing	Partial	Method recorded only; no gateway
Persistent database	Partial	Schemas exist; controllers use MemoryStore
Delivery-partner tracking; admin console	Not found	Status timeline only; no admin page

RESULTS AND DISCUSSION

A. Functional results

The back end started without a database and served all of its routes. With the default Tiptur location, all ten seeded shops lie within 1.6 km. A radius of 0.7 km returns four shops (0, 0.6, 0.6 and 0.7 km) and a radius of 0.1 km returns only the shop at the location, in ascending order of distance. The thirty seeded products were filtered correctly by category, price, rating, stock and text, and sorted correctly by price.

An order with products from two shops produced two orders, one per shop. For an order of 12,998 rupees, delivery was free, the discount was 150 rupees and the grand total was 12,848 rupees, as Equation (3) requires; stock fell from 8 to 6 for the ordered product. A client that had joined an order room received the order:statusUpdated event when the seller changed the status. For the demo customer, who prefers Cricket, Badminton and Running, the top three recommendations were Cricket and Badminton products with equal scores of 134, which is consistent with the scoring rules.

B. Test results

All 15 unit tests passed. Of the 59 integration tests, 58 passed and one failed (Figure 3), giving 73 passes in 74 tests. The failed test, T49, checks that a customer cannot change the status of an order; the server returned status

200 and accepted the change. This is a defect of the delivered code (a missing role check on PUT /api/orders/:id/status), not an error of the test. Table 5 lists this and the other defects found by the tests and by direct requests; the severity column is the team's suggested rating, not a measured value.

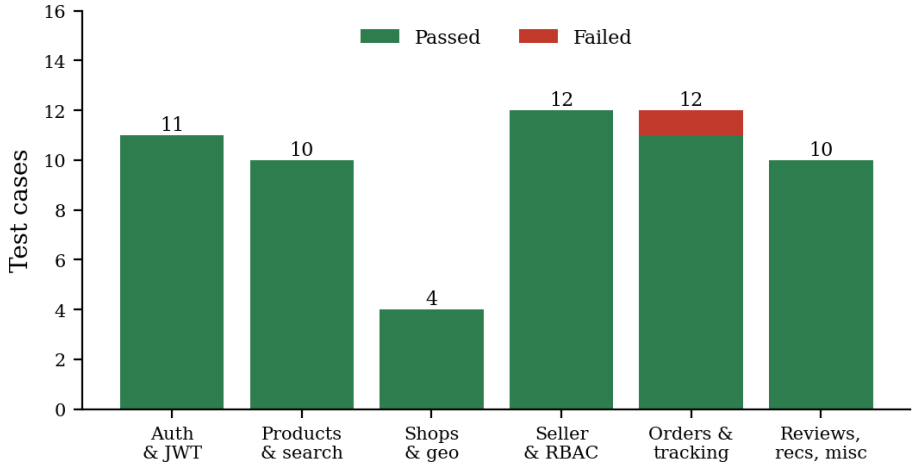


Figure 3: Integration test results by group (58 passed, 1 failed).

Table 5: Defects and weaknesses identified during testing.

ID	Finding	Severity	Suggested correction
D-1	Customer token accepted by PUT /api/orders/:id/status (T49)	High	Restrict to sellers of the order's shop
D-2	Registration accepts any role from the request body	High	Always create customers
D-3	Fixed password accepted for any existing account	High	Remove demonstration shortcut
D-4	Default JWT secret in source code	Medium	Require JWT_SECRET at start-up
D-5	Socket.io rooms joined without authentication	Medium	Verify token in handshake
D-6	Delivery-fee boundary differs between cart and server	Low	Use one shared pricing function
D-7	Distance function returns 1.2 km when a coordinate is 0	Low	Test for null, not falsy values
D-8	Product rating not recalculated after a review	Low	Update rating in review controller

C. Performance

Table 6 and Figure 4 give the measured response times. Read operations averaged about 1.7 to 3.2 ms. Login was much slower (about 110 ms) because it performs a bcrypt comparison with cost factor 10; this deliberate cost makes password guessing expensive [10]. The figures show the cost of the application code alone on a small data set. They exclude network delay, browser rendering, external map tiles and images, and concurrent users, none of which was tested.

Table 6: Measured response times (ms, sequential requests).

Operation	Requests	Mean	Median	95th pct.
GET /api/health	200	2.98	1.97	7.52
GET /api/products (30 items)	200	3.16	2.31	7.59
GET /api/products?category=Cricket&sort=price_asc	200	1.74	1.25	4.30
GET /api/products?search=racket	200	1.67	1.08	4.31
GET /api/products/prod-1	200	2.01	1.53	4.15
GET /api/shops/nearby (15 km)	200	2.30	1.54	6.36
GET /api/recommendations	200	2.57	1.91	5.90
POST /api/auth/login (bcrypt)	50	110.42	113.20	138.71
GET /api/auth/me (JWT verify)	200	2.29	1.57	6.42

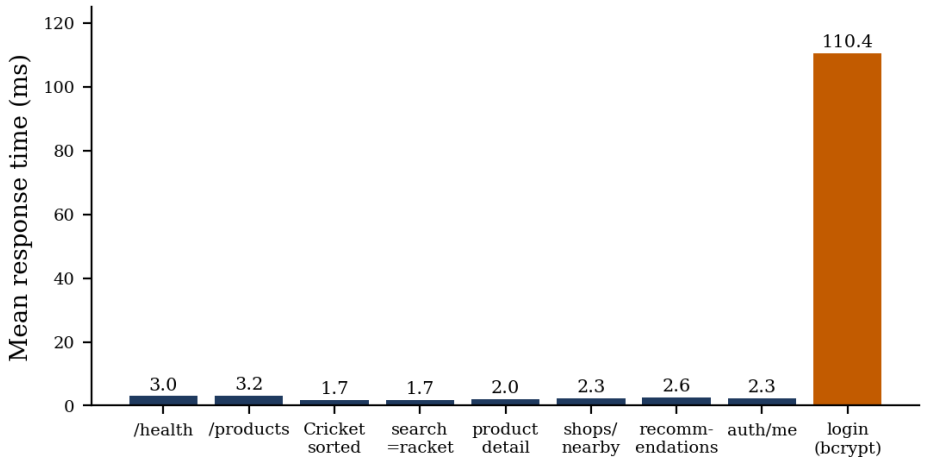


Figure 4: Mean response time of the main API operations.

D. Discussion

The results show that the main design decisions work as intended: filtering and distance computation are fast at this data size, orders split correctly by shop, real-time events arrive without reloading, and role checks protect most seller operations. The evaluation also separates the prototype from a deployable system. The rule-based recommender, although labelled as "AI" in the interface, is a transparent set of weights, and its quality has not been evaluated against user behaviour, so no accuracy figure is reported. Linear scans over in-memory arrays are negligible for ten shops and thirty products but would need indexed queries at larger scale; the declared 2dsphere and text indexes would serve that purpose [19].

LIMITATIONS

1. Data is held in memory and is lost on restart; the Mongoose schemas are not used by the controllers.
2. Payment is recorded but not processed; the payment status is set to Completed for UPI and card without verification.
3. Several security shortcuts remain: a universal demonstration password, unrestricted role selection at registration, an order-status route without a role check, a default JWT secret and unauthenticated Socket.io room joins. These follow the pattern of broken access control that OWASP lists as a major web-application risk [20].
4. The seller dashboard contains fixed or simulated values, and the product rating is not recalculated when a review is added.
5. The user interface was verified by reading its source code and was not run in the test environment; manual interface test cases are still to be executed.
6. Performance was measured for sequential requests on one single-core machine and says nothing about concurrency, a database or other hardware.

CONCLUSIONS AND FUTURE SCOPE

This paper presented SportKart, a hyperlocal sports e-commerce platform for small-town retail. It combines Haversine-based shop discovery, product search and comparison, orders split by shop, Socket.io real-time tracking, a seller console and rule-based recommendations in a Next.js and Node.js application that runs without paid services. Of 74 tests, 73 passed, and read requests completed in a few milliseconds on a single-core machine.

Future work should replace the in-memory store with the existing Mongoose models and use the 2dsphere and text indexes; remove the demonstration password and restrict role selection, add role and ownership checks to the order-status route, and authenticate Socket.io connections; integrate a payment gateway with server-side verification; compute dashboard values from stored orders; record browsing events and evaluate the recommender on real interaction data; add delivery-partner tracking and a shop-verification console; and convert the tests of this paper into an automated suite with end-to-end and load testing. Onboarding of shops in Tumakuru, Hassan and Arsikere would then test the platform beyond its seed data.

ACKNOWLEDGEMENTS

The authors express their sincere gratitude to their guide, Shashidhara M S, Associate professor, Department of Computer Science and Engineering, Kalpataru Institute of Technology, Tiptur, for valuable guidance and constant support throughout this work. They also thank their families and friends for their support.

REFERENCES

[1] K. Schade, M. Hübscher and T. Korzer, “Smart retail in smart cities: Best practice analysis of local online platforms,” in Proc. 15th Int. Joint Conf. e-Business and Telecommunications (ICETE 2018), vol. 1, 2018, pp. 147–157.

[2] J. Hong, H. Zo and H. Jo, “Digital commerce goes local: An empirical analysis of transactional behaviors in hyperlocal e-commerce,” Int. J. Electron. Commerce, vol. 30, no. 1, pp. 117–146, 2026.

[3] P. Resnick and H. R. Varian, “Recommender systems,” Commun. ACM, vol. 40, no. 3, pp. 56–58, Mar. 1997.

[4] J. B. Schafer, J. A. Konstan and J. Riedl, “E-commerce recommendation applications,” Data Mining Knowl. Discovery, vol. 5, no. 1–2, pp. 115–153, 2001.

[5] B. Sarwar, G. Karypis, J. Konstan and J. Riedl, “Item-based collaborative filtering recommendation algorithms,” in Proc. 10th Int. Conf. World Wide Web (WWW), 2001, pp. 285–295.

[6] G. Linden, B. Smith and J. York, “Amazon.com recommendations: Item-to-item collaborative filtering,” IEEE Internet Comput., vol. 7, no. 1, pp. 76–80, Jan./Feb. 2003.

[7] G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions,” IEEE Trans. Knowl. Data Eng., vol. 17, no. 6, pp. 734–749, Jun. 2005.

[8] R. Burke, “Hybrid recommender systems: Survey and experiments,” User Model. User-Adapted Interaction, vol. 12, no. 4, pp. 331–370, 2002.

[9] R. W. Sinnott, “Virtues of the haversine,” Sky and Telescope, vol. 68, no. 2, p. 159, 1984.

[10] N. Provos and D. Mazières, “A future-adaptable password scheme,” in Proc. USENIX Annu. Tech. Conf., 1999, pp. 81–91.

[11] M. Jones, J. Bradley and N. Sakimura, “JSON Web Token (JWT),” IETF, RFC 7519, May 2015.

[12] I. Fette and A. Melnikov, “The WebSocket protocol,” IETF, RFC 6455, Dec. 2011.

[13] Socket.IO Documentation, Version 4. [Online]. Available: https://socket.io/docs/v4/

[14] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, Univ. California, Irvine, CA, USA, 2000.

[15] R. S. Pressman and B. R. Maxim, Software Engineering: A Practitioner’s Approach, 9th ed. New York, NY, USA: McGraw-Hill, 2019.

[16] Next.js Documentation. [Online]. Available: https://nextjs.org/docs

[17] Leaflet API Reference. [Online]. Available: https://leafletjs.com/reference.html

[18] OpenStreetMap Foundation, OpenStreetMap. [Online]. Available: https://www.openstreetmap.org

[19] MongoDB Manual: Geospatial Queriesand 2dsphere Indexes. [Online]. Available: https://www.mongodb.com/docs/manual/geospatial-queries/

[20] OWASP Foundation, OWASP Top 10:2021. [Online]. Available: https://owasp.org/Top10/