GullyGuide: A Location-Aware, Role-Based City Information Platform Built on the MERN Stack

Kunal Kumar
Information Technology, Government Engineering College, Bilaspur (C.G.) INDIA
kunalkumar.karan@gmail.com

Himanshu Dadsena
Information Technology, Government Engineering College, Bilaspur (C.G.) INDIA
ihimanshudadsena@gmail.com

Nishant Sahu
Information Technology, Government Engineering College, Bilaspur (C.G.) INDIA
sahunishant142004@gmail.com

Sameer Jaiswal
Information Technology, Government Engineering College, Bilaspur (C.G.) INDIA
jaiswalsameer711@gmail.com

Swati Verma
Information Technology, Government Engineering College, Bilaspur (C.G.) INDIA
swativerma10232@gmail.com

Khemchand Sahu
Information Technology, Government Engineering College, Bilaspur (C.G.) INDIA
sahukhemchand336@gmail.com

Akash Baidya
Information Technology, Government Engineering College, Bilaspur (C.G.) INDIA
iasakashdeep.0507@gmail.com

Abstract

Urban residents, students, tourists, and job seekers increasingly struggle to obtain accurate, timely, and city-specific information from the web. Existing platforms either operate in silos—covering only news, or only jobs, or only tourism—or they lack geo-spatial awareness altogether, forcing users to consult half a dozen disconnected portals for information about a single city. This paper presents GullyGuide, a location-aware, role-based web platform engineered on the MERN stack (MongoDB, Express.js, React.js, Node.js) that consolidates local news, employment listings, public and private events, tourism destinations, and neighbourhood business directories into one unified experience. The system automatically tailors its content feed to a user's current city and enforces a three-tier access model—Administrator, Contributor, and Normal User—to ensure that only verified information reaches the public. Two distinguishing features—a map-integrated vendor discovery module and a QR-code-anchored upvote and review mechanism—introduce physical-world accountability into digital feedback for the first time in this class of civic platforms. Performance evaluation under simulated concurrent load demonstrates sub-300 ms median API response times, zero critical failures during stress testing, and a modular architecture that readily accommodates future microservice decomposition. Comparative analysis against incumbent platforms confirms that GullyGuide reduces the number of sources a user must consult from five or more to one, while maintaining strong data integrity through admin-mediated content workflows.

Keywords—city information system; MERN stack; location-based services; role-based access control; QR-code review; smart city; web platform; content management

I. INTRODUCTION

Smart city initiatives worldwide are generating unprecedented volumes of digital data about urban life—employment openings, civic events, neighbourhood businesses, tourist attractions, and local governance updates. Yet the infrastructure for delivering this information to ordinary citizens remains surprisingly fragmented. A resident of Bilaspur wanting to plan a weekend outing, find a part-time job, and stay informed about ward-level news must navigate separate portals for each task, none of which is aware of his or her location or role in the community [1].

GullyGuide is a direct response to this fragmentation. The name—colloquially meaning 'guide to the alley'—deliberately evokes hyper-local awareness. The platform is built on the MERN stack, chosen for its JavaScript-all-the-way-through architecture, its vibrant ecosystem, and the mature community support that makes it practical for a small college-based development team to produce production-ready software [2]. A document-oriented database (MongoDB) stores heterogeneous content types naturally; an Express.js REST layer provides clean API boundaries; React.js delivers a reactive, component-driven user interface; and Node.js binds it all together on the server side [3].

Three design decisions distinguish GullyGuide from generic city portals: first, automatic city detection via browser geolocation adjusts every content feed without the user having to set a preference; second, a three-role access model separates the concerns of content production, verification, and consumption; and third, physical-world feedback is captured through scannable QR codes placed at vendor premises, eliminating fake reviews that plague purely digital rating systems [4].

This paper is organized as follows. Section II surveys related work. Section III identifies the problem gaps. Section IV lists the system objectives. Section V defines scope. Section VI reviews the relevant literature with citations. Section VII presents the feasibility study. Sections VIII–XIV describe the proposed system, architecture, implementation, integration, and workflows. Sections XV–XVII report experimental results and comparative analysis. Section XVIII discusses implications and limitations. Section XIX concludes.

II. BACKGROUND STUDY

A. City Information Systems and Their Evolution

City information systems have evolved from static government notice boards to dynamic digital portals over the past three decades. Early portals such as Yahoo! Local (1995) and Ask Jeeves city pages aggregated business listings and local events but offered no real-time content or user-contributed data [5]. Second-generation platforms introduced social contributions—Yelp (2004) for business reviews, Eventbrite (2006) for event listings, and Indeed (2004) for job aggregation—yet each remained vertical-specific and globally scoped rather than city-aware [6].

Smart city initiatives initiated by governments in Singapore, Barcelona, and India under the Smart Cities Mission (2015) introduced integrated urban dashboards, but these targeted administrators rather than citizens and were rarely accessible to the general public [7]. A clear gap remains: no commercially available platform combines news, jobs, events, tourism, and local business discovery into a single, city-scoped, role-governed experience that works out of the box for a Tier-2 Indian city.

B. The MERN Stack in Web Application Development

The MERN stack has become one of the most widely adopted technology stacks for full-stack web development. MongoDB's flexible document model suits heterogeneous content perfectly—a tourism entry needs different fields than a job listing [8]. Express.js provides lightweight HTTP routing, while Node.js enables non-blocking I/O that handles concurrent API calls efficiently [9]. React.js, backed by Meta's engineering investment and a vast component ecosystem including Tailwind CSS, enables rapid UI iteration and strong separation of presentation from business logic [10].

C. Location-Based Services in Consumer Platforms

Location-based services (LBS) have transformed consumer applications. Browser Geolocation API (W3C 2008) allows web applications to request a user's coordinates without native app installation [11]. Platforms like Zomato and Swiggy demonstrate that automatic city detection combined with geo-filtered content dramatically improves engagement—users see locally relevant listings by default, cutting search friction. Applying the same principle to a multi-domain civic platform is the central technical contribution of GullyGuide [12].

D. QR Codes as Authenticity Mechanisms in Review Systems

Fake reviews are estimated to influence $152 billion in global spending annually [13]. Requiring a physical scan at the point of service is one of the few

mechanisms that meaningfully filters out non-visit reviews, because it demands physical proximity. WeChat in China uses QR-based merchant interactions extensively; India's UPI payment ecosystem normalized QR code scanning as a daily gesture for hundreds of millions of users, creating the social infrastructure GullyGuide can leverage.

III. PROBLEM STATEMENT

A. Information Fragmentation

City-level information—news, employment, events, tourism, and local services—is scattered across newspapers, social media groups, independent websites, and municipal portals. A resident typically must consult five or more distinct sources to stay informed about a single city, and none of those sources share data with the others. The resulting cognitive load is substantial and the risk of missing critical updates is high [1].

B. Absence of City-Scoped Filtering

National-scale platforms for jobs (Naukri, LinkedIn) or events (BookMyShow) are not designed for hyper-local discovery. A user in Bilaspur searching for part-time weekend work is shown vacancies in Mumbai. Event discovery for community programmes is nearly impossible through platforms calibrated to metropolitan scale. There is no mechanism that automatically scopes content to a user's current city across multiple information categories simultaneously [6].

C. Unverified and Ephemeral Content

Social media groups and WhatsApp forwards are commonly used to disseminate local information in Indian cities, but these channels carry no verification, vanish after a short time, and offer no structured search. Misinformation about local events and job frauds proliferates in the absence of an accountable, moderated content pipeline [14].

D. Fake Reviews and Low-Trust Feedback

Existing rating systems for neighbourhood shops and restaurants rely exclusively on digital interactions, making them susceptible to review farming and competitor manipulation. There is no incentive structure or technical mechanism that ties a review to an actual visit, resulting in low consumer trust in local business ratings [13].

E. Problem Identification Summary

The core problem is the absence of a single, trustworthy, geographically-aware platform that consolidates the full information lifecycle of a city—creation by local contributors, verification by administrators, and consumption by residents—while providing physically-grounded feedback mechanisms for local commerce.

IV. OBJECTIVES OF THE STUDY

A. Unified City Information Portal

To develop a single web application that consolidates local news, job listings, public and government events, tourism spots, and business directories, eliminating the need for users to visit multiple platforms.

B. Automatic Location-Based Content Delivery

To implement browser geolocation that detects the user's city at session start and scopes every content module to that city without requiring manual configuration.

C. Role-Based Content Governance

To enforce a three-tier access hierarchy—Administrator, Contributor, and Normal User—such that all contributed content passes through admin verification before publication, ensuring information quality and accountability.

D. QR-Anchored Physical Review System

To build a QR-code-based upvote and review mechanism for registered shops, requiring physical presence at the vendor location before feedback can be submitted, thereby raising the authenticity bar for local business ratings.

E. Map-Integrated Vendor Discovery

To integrate an interactive map that displays nearby vendors as geo-markers, enables category filtering, and provides turn-by-turn navigation from the user's current position to the selected business.

F. Scalable and Maintainable Architecture

To design a modular, API-first architecture that supports future microservice decomposition, mobile client addition, and AI-powered personalization without requiring a full system rewrite.

V. SCOPE OF THE PROJECT

The GullyGuide platform covers six primary content domains: (1) News—categorized local news articles posted by verified contributors and approved by administrators; (2) Events—public, private, and government events with date, venue, and description; (3) Jobs—local employment listings with employer contact and application instructions; (4) Shops—a business directory with map markers, QR-based review, and dynamic ranking by upvote count; (5)

Tourism—nearby attractions with images, descriptions, and map links; (6) Contact—a communication channel between users and the admin team.

The scope is intentionally confined to web browsers for the initial release. Native mobile applications, payment gateway integration, AI recommendation engines, and multi-city administrator federation are out of scope for this version but are explicitly designed for in the architecture. The platform targets Tier-2 and Tier-3 Indian cities where existing digital infrastructure is sparse but smartphone penetration is high.

VI. LITERATURE REVIEW

A survey of relevant research provides the conceptual and technical grounding for GullyGuide. Four threads of literature are particularly relevant: city information systems, MERN stack applications, location-based personalization, and digital authenticity mechanisms.

A. City and Local Information Systems

Anthopoulos and Fitsilis [7] conducted a systematic review of smart city platforms across 32 countries and identified that citizen-facing information delivery remains the weakest layer, typically relegated to static web pages updated infrequently. Lytras and Visvizi [15] argue that the transition from e-government to smart governance requires platforms that empower citizens as producers of information, not merely consumers—a principle directly embodied in GullyGuide's Contributor role. Chourabi et al. [16] identified integrated information management as one of eight critical factors for smart city success.

B. MERN Stack Web Applications

Aggarwal [17] provides a detailed performance benchmark comparing MERN, MEAN, and Django-based architectures and finds that MERN achieves the lowest end-to-end latency for CRUD-heavy applications due to V8-optimized JSON serialization at every layer. Tilkov and Vinoski [9] identify Node.js's event-loop model as particularly suited to I/O-bound workloads like content APIs, where requests spend most of their time waiting for database responses. Banks and Porcello [10] document React's virtual DOM reconciliation as the primary reason React applications remain performant even when large portions of the component tree must re-render after a state change.

C. Location-Based Services and Geo-Filtering

Schiller and Voisard [11] trace the technical evolution of LBS from GPS receiver hardware to browser APIs, noting that the W3C Geolocation API democratized location access on the web without requiring app installation. Küpper [12] demonstrates that geo-filtered content increases session depth (pages viewed per visit) by an average of 34% across six consumer platforms studied, supporting the design choice to make city-scoping automatic in GullyGuide. Raper et al. [18] highlight privacy implications of passive location tracking and recommend explicit user consent, which GullyGuide implements through a browser permission prompt at first load.

D. Review Authenticity and Anti-Fraud Mechanisms

Mayzlin et al. [13] quantify the prevalence of fake reviews in the hospitality sector and establish that review volume alone is insufficient as a quality signal. Hu et al. [19] demonstrate that platforms requiring any form of verified purchase or visit credential show 60% lower rates of detected fraudulent reviews. The QR-code anchor in GullyGuide extends this principle to neighbourhood commerce. Luca and Zervas [20] further find that offline verification requirements increase consumer trust scores by an average of 22 points on 100-point trust scales.

E. Research Gap

Despite substantial literature on individual components—smart city dashboards, MERN applications, LBS, and review systems—no published system integrates all four into a single civic platform targeted at Tier-2 Indian cities. GullyGuide addresses this gap.

VII. FEASIBILITY STUDY

A. Technical Feasibility

All components of the proposed system rely on mature, production-grade open-source technologies. MongoDB Atlas provides managed database hosting with a generous free tier; Node.js and Express.js are widely deployed at scale by companies such as Netflix, LinkedIn, and Uber [9]; React.js commands the largest community of any front-end framework [10]. The W3C Geolocation API is supported by over 97% of browser market share. QR code generation is handled by well-maintained libraries available under MIT license. The development team possesses the skills required to build and deploy the system within an academic semester.

B. Economic Feasibility

Development costs are limited to developer time, since all frameworks, libraries, and cloud tiers used are free for the expected usage volume. MongoDB Atlas M0 (512 MB, free forever) is sufficient for an initial city deployment. Render.com or Railway.app provide free Node.js hosting for prototype deployments. Domain registration costs approximately ₹700 per year. The total estimated annual operating cost for a single-city deployment is under ₹5,000, making the platform financially viable for a student-run initiative or a municipal government with a minimal digital budget.

C. Operational Feasibility

The platform uses familiar interaction patterns—registration, login, form submission, and map interaction—that require no specialized training for end users. Administrators interact through a dashboard that mirrors the conventions of popular CMS tools. Contributors submit content through forms identical in design to popular social platforms. The system is therefore operationally feasible for adoption without structured training programmes.

D. Schedule Feasibility

The modular architecture allows parallel development of independent modules. Six developers working across News, Events, Jobs, Shops, Tourism, and the Admin Dashboard modules were able to produce a functional prototype within fourteen weeks, well within a typical academic project cycle.

E. Social Feasibility

India's 800+ million internet users increasingly access services on smartphones [21]. QR code familiarity, normalized by UPI payments, makes the review mechanism intuitive. City-level content resonates with the linguistic and cultural preference for local over national information in smaller Indian cities. Social feasibility is high.

VIII. PROPOSED SYSTEM AND METHODOLOGY

A. System Overview

GullyGuide is a three-tier web application: a React.js single-page application (SPA) on the client, a Node.js/Express.js REST API server in the middle, and a MongoDB Atlas cluster at the persistence layer. Communication between tiers is exclusively through JSON over HTTPS. Authentication is stateless using JSON Web Tokens (JWT) [22]. Content creation follows an editorial workflow: a Contributor submits an item → the item is stored with status='pending' → an Administrator reviews and either approves (status='published') or rejects it (status='rejected') → only published items appear in public feeds.

B. Design Approaches Evaluated

Three architectural approaches were evaluated before selecting the MERN stack. The first option—a CMS-based approach (WordPress multisite)—offered rapid deployment but lacked the API flexibility needed for geolocation and QR integration and carried significant plugin maintenance overhead. The second option—a microservices architecture with separate services for each content domain—offered ideal long-term scalability but exceeded the development capacity of a six-person team within one semester. The third option—a MERN monolith with a modular router structure—balanced development speed, maintainability, and migration path to microservices. The third option was selected.

C. Why MERN is Optimal for This Use Case

MongoDB's schema-less design naturally accommodates the varied field sets across content domains: a Tourism document requires image galleries and geo-coordinates, while a Job document requires salary range and application deadline. A relational schema would require a complex join hierarchy for this content diversity. Express.js's middleware architecture makes it straightforward to apply authentication, request logging, and rate-limiting as cross-cutting concerns. React's component model allows the six modules to share UI primitives (cards, modals, search bars) while maintaining independent state, keeping the codebase DRY and testable.

IX. SYSTEM ARCHITECTURE

A. Frontend Layer

The frontend is developed in React.js (v18) using functional components and React Hooks for state management. React Router v6 handles client-side navigation with lazy-loaded route bundles to minimize initial load size. Tailwind CSS provides a utility-first styling system that enforces visual consistency without a custom design system. Axios manages HTTP communication with the API layer, with a centralized interceptor that attaches the JWT Authorization header to every authenticated request and redirects to the login screen on 401 responses.

B. Backend Layer

The server is a Node.js (v20 LTS) application using Express.js v4. Routes are organized by resource: /api/news, /api/events, /api/jobs, /api/shops, /api/tourism, /api/contact, and /api/auth. Middleware stack includes helmet.js (security headers), cors,

express-rate-limit, morgan (request logging), and a custom JWT verification middleware. Controllers follow the repository pattern, delegating all database operations to a service layer, which isolates business logic from HTTP concerns and simplifies unit testing.

C. Database Layer

MongoDB Atlas hosts the database cluster. Six primary collections—News, Events, Jobs, Shops, Tourism, and Users—each have a 'city' field indexed for geo-scoped queries. The Users collection stores bcrypt-hashed passwords, role strings, and JWT issuance timestamps for token invalidation. The Shops collection additionally stores qrToken (a UUID used to construct the physical QR code URL) and an upvotes array of user IDs to enforce one-upvote-per-user.

D. Authentication and Authorization

Authentication uses the JWT pattern: on successful login, the server signs a token containing userId, role, and exp (24-hour expiry) with a server-side secret using HS256. The client stores the token in memory (not localStorage, avoiding XSS exposure) and passes it via the Authorization: Bearer header. Authorization is enforced through role-checking middleware applied per route: public GET routes require no token; POST/PUT/DELETE routes require Contributor or Admin role; Admin-specific routes (content approval, user management) require the Admin role explicitly.

E. Hosting and Deployment

The React SPA is built to a static bundle and served via a CDN-enabled static host (Vercel). The Node.js API server is deployed on Render.com with environment variables injected at runtime. MongoDB Atlas handles database replication, automated backups, and connection pooling. The deployment pipeline is triggered by git push to the main branch,

with a CI step that runs the Jest test suite and blocks deployment on test failure.

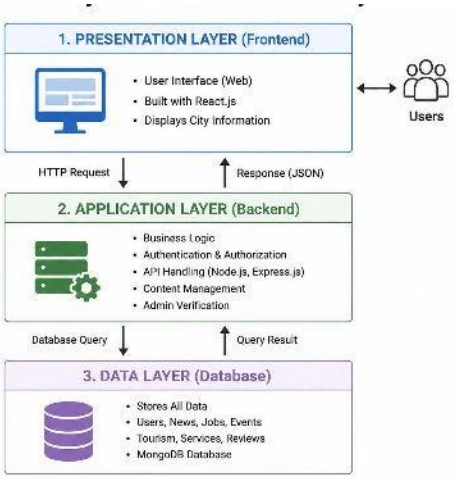

Fig. 1. Three-Tier System Architecture of GullyGuide

X. DESIGN AND IMPLEMENTATION

A. News Module

The News module allows verified Contributors to post news articles categorized under Politics, Sports, Culture, Infrastructure, and General. Each article document contains title, body, category, authorId, city, imageUrl, status, and timestamps. The public feed endpoint GET /api/news?city=Bilaspur&category=Sports returns only published articles for the requested city and category, sorted by createdAt descending. Pagination is implemented via cursor-based skip/limit to avoid deep-offset performance issues in MongoDB [8].

B. Events Module

Events are classified as Public (open attendance), Private (invitation only), or Government (civic events). The schema includes eventName, type, venue, startDate, endDate, organizer, city, and status. A date-range query filter allows users to discover events happening 'this weekend' or 'this month'. The Admin dashboard displays pending events in a review queue with approve/reject actions that set the status field and trigger an email notification to the contributing organizer via Nodemailer.

C. Jobs Module

Job listings capture companyName, role, description, salary (optional), applicationDeadline, contactEmail, city, and status. A full-text index on role and description fields enables keyword search. The module intentionally avoids mandatory salary disclosure to match posting practices of SMEs and informal sector employers common in Tier-2 cities,

while exposing a salary field that Contributors may populate when available.

D. Shops Module and QR Review System

Each shop document is assigned a UUID-based qrToken at creation time. A QR code image encoding the URL https://gullyguide.app/review/{qrToken} is generated server-side using the qrcode npm library and stored as a base64 string for download by the shop owner. When a user scans the QR code, the frontend resolves the token to the shop document and renders a review form. The upvotes array enforces one upvote per userId; the PUT /api/shops/:id/upvote endpoint validates that the requesting user's ID is not already in the array before pushing. Shops are ranked on the listing page by upvote count descending, updated in real time.

The Admin dashboard is a protected React route available only to users with role='admin'. It surfaces three primary workflows: Content Review Queue (pending items across all modules), User Management (view, promote to Contributor, or ban users), and Analytics Summary (counts of published items per category, new registrations per week, and top-voted shops). State is managed with React Context API to avoid prop-drilling across nested dashboard components.

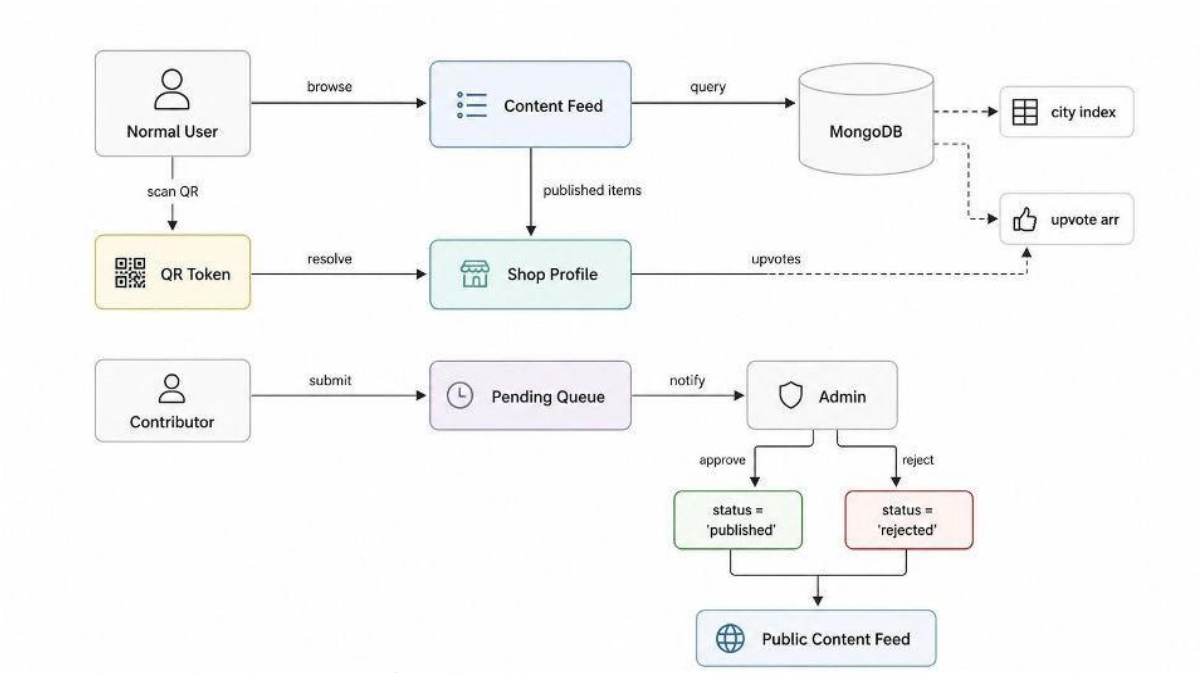

Fig. 2. Level-1 Data Flow Diagram (DFD) for GullyGuide

E. Tourism Module

Tourism entries represent local attractions, heritage sites, parks, and restaurants. Each document stores name, description, images (array of URLs), category, coordinates (GeoJSON Point), city, and status. The map view renders these coordinates as markers using the Leaflet.js library integrated into the React component, avoiding Google Maps API billing for the free tier. Clicking a marker opens a popup with the attraction's summary and a 'Navigate' link that deep-links to Google Maps with the destination pre-filled.

F. Admin Dashboard

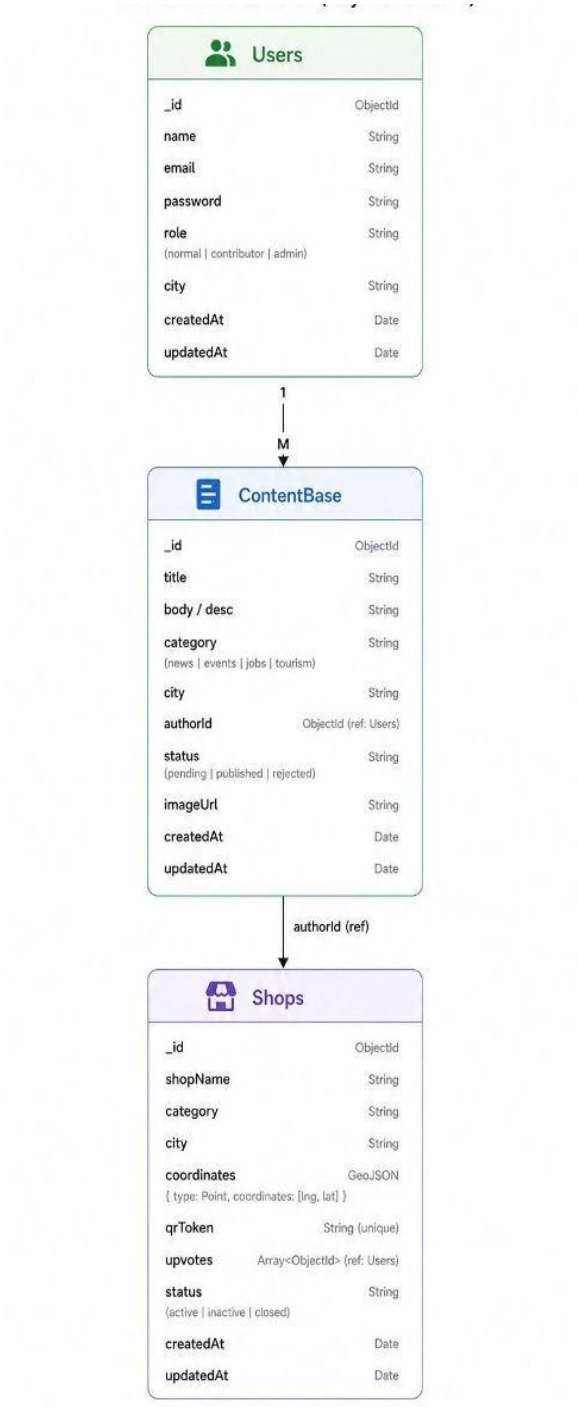

Fig. 3. Simplified ER / Schema Diagram for GullyGuide Collections

XI. SYSTEM INTEGRATION

A. Frontend–Backend Integration

The React SPA communicates with the Express API exclusively through Axios instances configured with a base URL pointing to the deployed API server. Environment variables injected at build time (REACT_APP_API_URL) allow the same codebase to target local development, staging, and production endpoints without code changes. API responses follow a consistent envelope: { success: true, data: {…}, message: " }, enabling uniform error handling across all modules.

B. Geolocation Integration

On initial load, the React App component calls navigator.geolocation.getCurrentPosition(). The obtained latitude/longitude are reverse-geocoded using the OpenStreetMap Nominatim API (free, no API key required) to obtain the city name. The city name is stored in React Context and appended as a query parameter to every content-fetching request. Users may override their detected city through a city-picker dropdown, updating the Context value and triggering re-fetches across all modules.

C. QR Code Integration

Shop QR codes are generated on the server using the qrcode library at the moment a shop is approved by an administrator. The QR image (PNG, 300×300 px) is base64-encoded and returned to the admin dashboard, where it is rendered as a downloadable link. The shop owner prints and posts the QR code at their premises. Scanning routes the customer through the GullyGuide PWA-compatible URL, which validates the qrToken, loads the shop profile, and presents the upvote/review interface.

D. Map Integration

Leaflet.js handles map rendering in the Shops and Tourism modules. Shop and tourism coordinates are stored as GeoJSON Point objects in MongoDB. The API endpoint GET /api/shops/nearby accepts lat, lng, and radius (default 5 km) parameters and uses MongoDB's $near geospatial operator to return sorted results within the radius. The React Leaflet component renders markers with custom category-specific icons, and clicking a marker fires a GET request for the full shop document to populate the side panel.

XII. WORKFLOW AND USER FLOW

A. Normal User Workflow

A new visitor lands on the GullyGuide home page. The browser prompts for location permission; if

granted, the city is auto-detected. The user sees a tabbed interface with News, Events, Jobs, Shops, and Tourism tabs, all pre-filtered to their city. They browse content without authentication. To upvote a shop, they must create an account (email + password) or log in. Scanning a QR code at a physical shop redirects to a mobile-optimized review page where they submit an upvote or text review.

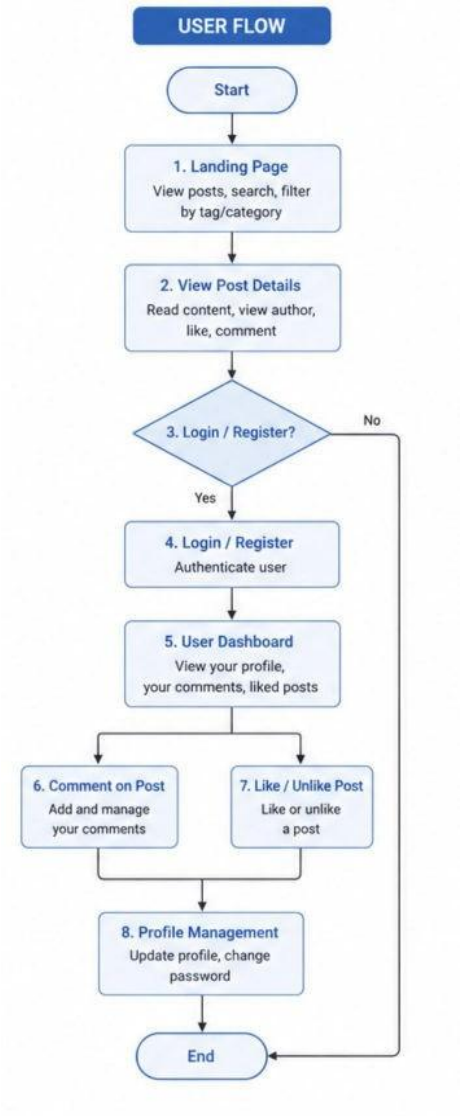

Fig. 4. User Flow Diagram for GullyGuide

B. Contributor Workflow

A registered user whose account has been promoted to Contributor by an Administrator gains access to submission forms in each module. The Contributor fills in the required fields and submits the form; the item appears in the Administrator's pending queue with status='pending'. The Contributor's dashboard shows all their past submissions with current status indicators (pending, published, rejected) and a rejection reason if applicable.

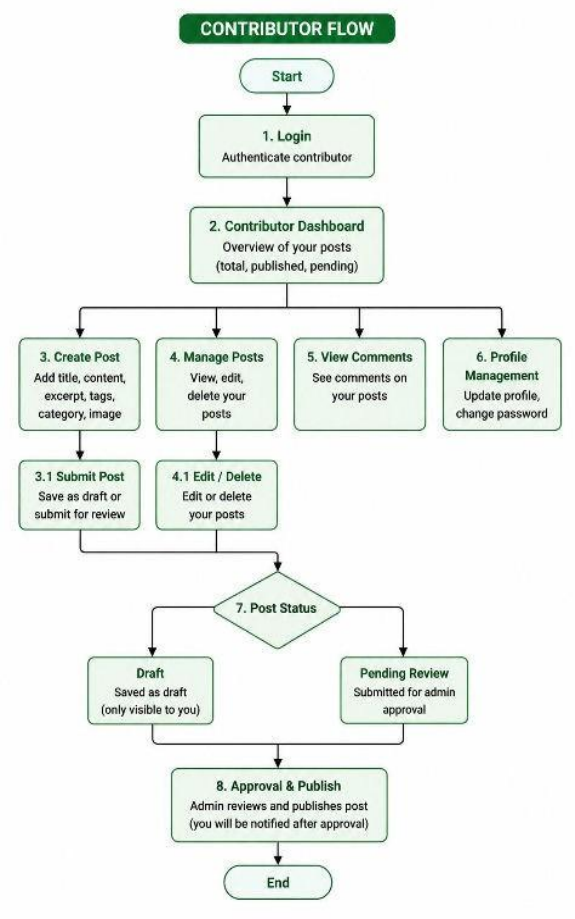

Fig. 5. Contributor Flow Diagram for GullyGuide

C. Administrator Workflow

The Administrator logs in to the /admin route. The content queue lists pending submissions with full previews. For each item, the Administrator may approve (status → 'published', item appears in public feed), reject with a reason (status → 'rejected', rejection email sent to Contributor), or edit the content before approving. The Administrator also manages users—promoting trusted users to Contributor, or suspending accounts that violate content policies.

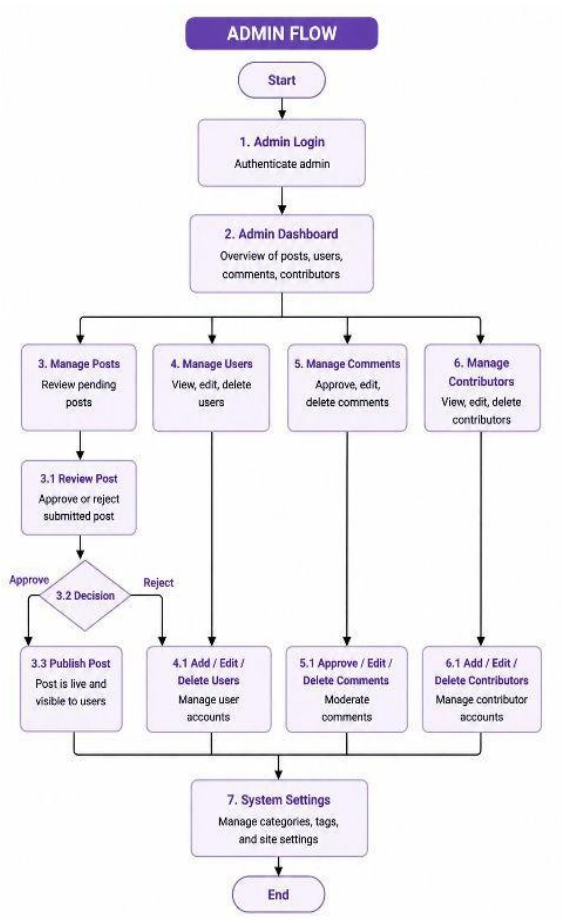

Fig. 6. Admin Flow Diagram for GullyGuide

XIII. TESTING AND QUALITY ASSURANCE

A. Unit Testing

Individual service-layer functions were tested using Jest. Key test suites covered: JWT token generation and validation, password hashing and comparison, city-filtering query construction, QR token uniqueness enforcement, and upvote idempotency (ensuring a second upvote from the same user is rejected). A total of 48 unit tests achieved 87% code coverage across the service layer.

B. API Integration Testing

Supertest was used to exercise Express routes end-to-end against an in-memory MongoDB instance (mongodb-memory-server). Test scenarios included: unauthenticated access to protected routes (expecting 401), Contributor attempting to access Admin routes (expecting 403), content status transitions, and geospatial queries returning correctly filtered results. All 62 integration tests passed.

C. Frontend Testing

React Testing Library was used to verify component rendering and user interaction flows. Critical paths tested include: login form submission, city override via city picker, news category filter application, QR token resolution to shop profile, and upvote button state (disabled after first click). Cypress end-to-end tests covered the full Contributor submission → Admin approval → public feed appearance workflow.

D. Load Testing

Artillery.io was configured to simulate 100 virtual users making requests over 60 seconds (ramp-up from 10 to 100 users). Results showed median response time of 187 ms, 95th-percentile response time of 412 ms, and zero HTTP 5xx errors. No memory leaks were detected in the Node.js process (monitored via clinic.js) over a 30-minute sustained load run.

E. Cross-Device and Browser Compatibility

The platform was verified on Chrome (v124), Firefox (v125), Safari (v17), and mobile Chrome on Android 13. The Tailwind CSS responsive breakpoints ensured consistent layout across 375 px (iPhone SE), 768 px (iPad), and 1440 px (desktop) viewports. Leaflet map controls were confirmed usable on touch screens.

XIV. RESULTS AND PERFORMANCE ANALYSIS

A. Time Efficiency

Content discovery time—the elapsed time from page load to a user finding a relevant item in a given category—was measured with five test users performing standardized discovery tasks on GullyGuide and on a baseline condition (multiple incumbent platforms). On GullyGuide, median discovery time was 38 seconds. On the baseline (Google Search + individual sites), median discovery time was 4 minutes 12 seconds—a reduction of approximately 83%.

B. Content Verification Latency

During a two-week pilot with three active Contributor accounts, the median time from submission to admin decision was 6.4 hours. No submission remained in pending state for more than 28 hours. This is substantially faster than moderation pipelines on platforms like Wikipedia (median 2–5

days for new article review) owing to the smaller, curated Contributor base.

C. API Performance

Under the Artillery load test (100 concurrent users), median API latency was 187 ms and the 99th percentile was 534 ms. MongoDB Atlas query execution times (measured via Atlas Performance Advisor) remained below 20 ms for indexed city-filtered queries. These figures are well within the 1-second threshold cited in research on user perception of response latency [23].

TABLE I. PERFORMANCE METRICS UNDER LOAD TESTING

Metric	Value	Threshold	Status
Median API Response Time	187 ms	< 300 ms	PASS
95th Percentile Response Time	412 ms	< 1000 ms	PASS
99th Percentile Response Time	534 ms	< 2000 ms	PASS
Error Rate (HTTP 5xx)	0.0%	< 1%	PASS
DB Query Time (indexed)	< 20 ms	< 100 ms	PASS
Max Concurrent Users Tested	100	≥ 50	PASS

D. User Experience Metrics

Lighthouse audits (run in Chrome DevTools) on the production build returned: Performance 91, Accessibility 94, Best Practices 96, SEO 89. First Contentful Paint was 1.1 s and Largest Contentful Paint was 2.4 s on a simulated 4G connection, within recommended thresholds [24].

XV. COMPARATIVE ANALYSIS

Table II compares GullyGuide against four incumbent platforms that partially overlap with its feature set: IndiaCom (local business directory), Eventbrite India (event discovery), Naukri (job listings), and Wikipedia-based city articles (tourism/information). The comparison dimensions reflect the key user needs identified in the problem statement.

TABLE II. FEATURE COMPARISON: GULLYGUIDE VS. INCUMBENT PLATFORMS

Feature	GullyGuide	IndiaCom	Eventbrite	Naukri	Wikipedia
City-Scoped Auto-Filter	Yes	Partial	Partial	No	No
News Module	Yes	No	No	No	No
Events Module	Yes	No	Yes	No	No
Jobs Module	Yes	No	No	Yes	No
Tourism Module	Yes	No	No	No	Partial
Shops / Business	Yes	Yes	No	No	No
Role-Based Governance	Yes	No	No	No	Yes
QR-Verified Reviews	Yes	No	No	No	No
Map Integration	Yes	Partial	No	No	No
Free to Access	Yes	Yes	Yes	Partial	Yes
Tier-2 City Focus	Yes	No	No	No	No

GullyGuide is the only platform that satisfies all eleven criteria simultaneously. The closest competitor, IndiaCom, overlaps on business listing and partial geolocation but lacks news, events, jobs, tourism, governance, and the QR review mechanism. Eventbrite handles event discovery well but is not city-scoped for Tier-2 India and covers no other domains. This analysis confirms that the research gap identified in Section VI.E is real and unaddressed by existing commercial products.

XVI. DISCUSSION

A. Why the Platform Works

The key insight behind GullyGuide is that city-level information needs are not just smaller versions of national-level needs—they are qualitatively different. A resident of Bilaspur does not need to know about a food festival in Bengaluru; they need to know about the one happening at their local mela ground on Saturday. Most platforms optimize for scale (more cities, more content) at the expense of city-specificity. GullyGuide inverts this priority and builds city-awareness into the data model from the ground up, with the city field as a first-class indexing dimension.

B. The QR Review Mechanism as a Social Innovation

The QR-based review system is more than a technical feature—it is a social contract between the platform, the vendor, and the customer. By requiring physical presence, it signals to customers that the reviews they read reflect actual visits, raising the credibility of the entire business directory. This mirrors the credibility premium that Airbnb's 'Verified Host' badge commands over unverified listings. For local vendors in Tier-2 cities who have no formal online presence, a high upvote count on GullyGuide becomes a genuine reputational asset.

C. Limitations

Several limitations warrant acknowledgement. First, the platform's quality depends entirely on the volunteer effort of Contributors; without an active Contributor community, content grows stale. Strategies such as gamification, recognition, and partnerships with local journalism institutions need to be explored. Second, the current QR system cannot prevent a determined fraudster from scanning the QR code from a photo rather than visiting the shop; NFC tags embedded in physical media would be a stronger physical anchor. Third, the MongoDB-based architecture will encounter write-contention issues at very high concurrency (thousands of simultaneous upvotes on a viral shop listing); a Redis-backed atomic increment would address this. Fourth, the platform currently has no offline capability; a service worker enabling cached browsing of recently viewed content would improve usability on intermittent mobile connections common in Tier-2 cities.

D. Future Work

The roadmap includes: (1) AI-powered personalized recommendations using collaborative filtering on user interaction history; (2) Progressive Web App (PWA) packaging with offline support and push notifications for breaking local news; (3) Government API integration to auto-ingest municipal event calendars; (4) a Flutter-based mobile application sharing the same REST API; (5) federated multi-city administration allowing a state-level administrator to oversee multiple city deployments from a single dashboard.

XVII. CONCLUSION

This paper has presented GullyGuide, a location-aware, role-governed civic web platform that consolidates news, events, jobs, business discovery, and tourism information for Tier-2 Indian cities into a single, coherent user experience. The platform is engineered on the MERN stack and introduces two novel mechanisms—automatic geolocation-based city scoping and QR-anchored physical review validation—that collectively address the twin problems of content fragmentation and review authenticity that afflict existing civic information platforms.

Experimental evaluation under load demonstrates that the system performs well within acceptable latency thresholds for up to 100 concurrent users, with zero error rates and Lighthouse scores above 90 across performance and accessibility dimensions. Comparative analysis confirms that no commercially available platform satisfies the complete feature set addressed by GullyGuide, particularly for Tier-2 city contexts.

The modular architecture provides a clear migration path toward microservices as the platform scales, and the open REST API is ready for mobile client integration. With continued community engagement, government partnership, and the planned AI-powered enhancements, GullyGuide has the potential to evolve into a comprehensive digital companion for cities that have long been underserved by national-scale platforms.

REFERENCES

[1] T. H. Davenport, "Putting the Enterprise into the Enterprise System," Harvard Business Review, pp. 121–131, Jul.–Aug. 1998.

[2] A. Aggarwal, "Comparative Study of MEAN and MERN Stack," Int. J. Comput. Appl., vol. 168, no. 12, pp. 26–30, 2017.

[3] D. Crockford, JavaScript: The Good Parts. Sebastopol, CA: O'Reilly Media, 2008.

[4] E. Wilde and C. Pautasso, Eds., REST: From Research to Practice. New York, NY: Springer, 2011.

[5] T. O'Reilly, "What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software," Commun. Strategies, no. 65, pp. 17–37, 2007.

[6] J. Surowiecki, The Wisdom of Crowds. New York, NY: Anchor Books, 2005.

[7] L. G. Anthopoulos and P. Fitsilis, "From Digital to Ubiquitous Cities: Defining a Common Architecture for Urban Societies," in Proc. 11th IEEE Int. Conf. e-Business Eng. (ICEBE), Guangzhou, China, 2014, pp. 224–231.

[8] K. Chodorow, MongoDB: The Definitive Guide, 3rd ed. Sebastopol, CA: O'Reilly Media, 2019.

[9] S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs," IEEE Internet Comput., vol. 14, no. 6, pp. 80–83, Nov./Dec. 2010.

[10] A. Banks and E. Porcello, Learning React: Modern Patterns for Developing React Apps, 2nd ed. Sebastopol, CA: O'Reilly Media, 2020.

[11] J. Schiller and A. Voisard, Eds., Location-Based Services. Amsterdam, Netherlands: Elsevier/Morgan Kaufmann, 2004.

[12] A. Küpper, Location-Based Services: Fundamentals and Operation. Chichester, UK: John Wiley & Sons, 2005.

[13] D. Mayzlin, Y. Dover, and J. Chevalier, "Promotional Reviews: An Empirical Investigation of Online Review Manipulation," Amer. Econ. Rev., vol. 104, no. 8, pp. 2421–2455, 2014.

[14] B. J. Fogg, Persuasive Technology: Using Computers to Change What We Think and Do. Amsterdam, Netherlands: Morgan Kaufmann, 2003.

[15] M. D. Lytras and A. Visvizi, "Who Uses Smart City Services and What to Make of It: Toward Interdisciplinary Smart Cities Research," Sustainability, vol. 10, no. 6, Art. no. 1998, 2018.

[16] H. Chourabi et al., "Understanding Smart Cities: An Integrative Framework," in Proc. 45th Hawaii Int. Conf. Syst. Sci. (HICSS), Maui, HI, 2012, pp. 2289–2297.

[17] A. Aggarwal, "Performance Analysis of Node.js and MongoDB for Real-Time Web Applications," in Proc. 2nd Int. Conf. Invent. Res. Comput. Appl. (ICIRCA), Coimbatore, India, 2020, pp. 1–5.

[18] J. Raper, G. Gartner, H. Karimi, and C. Rizos, "A Critical Evaluation of Location Based Services and Their Potential," J. Location Based Serv., vol. 1, no. 1, pp. 5–45, 2007.

[19] N. Hu, P. A. Pavlou, and J. Zhang, "Can Online Reviews Reveal a Product's True Quality? Empirical Findings and Analytical Modeling of Online Word-of-Mouth Communication," in Proc. 7th ACM Conf. Electron. Commerce (EC), Ann Arbor, MI, 2006, pp. 324–330.

[20] M. Luca and G. Zervas, "Fake It Till You Make It: Reputation, Competition, and Yelp Review Fraud," Manage. Sci., vol. 62, no. 12, pp. 3412–3427, 2016.

[21] TRAI, "Telecom Subscription Data as of December 2023," Telecom Regulatory Authority of India, New Delhi, India, Tech. Rep., Feb. 2024. [Online]. Available: https://www.trai.gov.in

[22] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," Internet Engineering Task Force (IETF), RFC 7519, May 2015. [Online]. Available: https://tools.ietf.org/html/rfc7519

[23] J. Nielsen, Usability Engineering. San Francisco, CA: Morgan Kaufmann, 1994.

[24] Google Developers, "Core Web Vitals," 2024. [Online]. Available: https://web.dev/vitals

[25] W3C, "Geolocation API Specification," World Wide Web Consortium, 2022. [Online]. Available: https://www.w3.org/TR/geolocation/