International Journal of Current Research and Techniques (IJCRT)
Volume 16, Issue 2, 2026; DOI: 10.5281/ZENODO.20406282
Original Research Article

MyColony: Design and Evaluation of a Colony-Based Digital Service Platform

Anurag Thakur, *B.E. CSE, Chitkara University, Dept. of Computer Science, Chitkara University, Punjab, India.*

Avinash Singh, *B.E. CSE, Chitkara University, Dept. of Computer Science, Chitkara University, Punjab, India.*

Harshit Satija, *B.E. CSE, Chitkara University, Dept. of Computer Science, Chitkara University, Punjab, India.*

Arnav Garg, *B.E. CSE, Chitkara University, Dept. of Computer Science, Chitkara University, Punjab, India.*

Manuscript Received: Apr 25, 2026; Revised: May 16, 2026; Published: May 27, 2026

Abstract: This paper describes MyColony, a web platform we built to help residents of housing colonies find and hire local service providers in a verified, colony-specific environment. To improve trust and accountability, the system integrates JWT-based authentication, role-based access control, and administrator-led service approval. MyColony follows a three-tier architecture consisting of a React.js frontend, a Node.js and Express.js backend, and a MongoDB database. Cloudinary is used for handling media uploads and image optimization. The platform supports service listing, colony- specific access, and administrative monitoring within a secure environment. Functional testing and system evaluation show that the platform effectively supports multi-colony service management, secure user interaction, and moderation workflows. Usage observations also indicate improved service discoverability and stronger community engagement within residential boundaries. The proposed system demonstrates how a structured digital platform can improve hyperlocal service exchange while maintaining community-level trust and governance.

Keywords: Hyperlocal Marketplace, Community Governance, MERN Stack, JWT Authentication, Three-Tier Architecture, Residential Colony, Service Moderation.

1. Introduction

Gated residential communities have become increasingly common in urban areas, changing how residents access everyday local services such as home maintenance, tutoring, food delivery, and repair work. Despite the growing demand for these services, the process of finding reliable providers still depends heavily on informal communication channels including WhatsApp groups, personal referrals, and neighborhood recommendations.

These methods are convenient but lack structure and accountability. Residents often have difficulty verifying service quality, comparing providers, or identifying trustworthy individuals. Research has shown that the absence of structured trust mechanisms discourages users from actively participating in localized service ecosystems [1]. In many cases, even reliable local providers remain undiscovered because information is scattered across informal networks.

Existing digital marketplaces such as UrbanClap, TaskRabbit, and other gig-based platforms are designed for large-scale operations rather than tightly connected residential communities. While these systems provide broad service access, they do not address colony-level concerns such as localized trust, administrative control, and restricted community participation. Trust within residential environments operates differently, where familiarity and accountability play a more significant role than open-market visibility.

The hyperlocal services sector continues to grow rapidly with increasing digital adoption and location-based service demand [9]. However, most existing systems focus only on individual components such as scheduling, ratings, or verification, without integrating governance and community- specific moderation into a single platform.

To address these limitations, this paper presents MyColony, a web-based hyperlocal service platform developed using the MERN stack. The system incorporates JWT-based authentication, role-based access control, and administrator- led moderation to ensure that only verified services are available within the community. During the development of the

platform, it was observed that residents preferred colony- specific listings because familiarity with local providers increased confidence during service selection.

A. Problem Definition

The primary challenge addressed in this research is the absence of a structured and trustworthy mechanism for discovering local services within residential colonies. Informal communication channels often become cluttered and difficult to manage, making service discovery inconsistent and unreliable. In addition, there is usually no centralized record of provider quality, previous interactions, or user feedback.

From the provider perspective, service visibility is equally limited. Reliable providers often struggle to reach residents effectively, while low-quality or unverified providers can continue operating without accountability.These limitations highlight the need for a community- focused platform that combines service discovery with trust, moderation, and localized governance. MyColony aims to address this challenge by providing a structured digital environment specifically designed for residential communities.

2. Literature Review

Recent research in hyperlocal digital systems has emphasized the importance of trust, verification, and community-focused service discovery. Although several platforms attempt to improve local service accessibility, most solutions address only specific aspects of the problem.

Previous studies on community-based digital marketplaces

[1] showed that restricting participation to verified members improves trust and reduces transaction friction. These systems mainly focused on verification and trust scoring mechanisms but provided limited support for administrative governance or scalability across multiple communities.

Chaudhary et al. [4] developed a household services platform that integrated real-time scheduling and transparent pricing. Their work highlighted the role of feedback systems in improving service quality. However, the platform lacked localized moderation mechanisms, making it difficult to enforce accountability within smaller residential environments.

Several location-based service platforms [3] focused on location-based service matching in underserved communities. The study demonstrated that even basic provider verification improved transaction completion rates and user confidence. However, the system was designed for a limited geographic region and lacked scalability for larger or multi-community deployments.

Some marketplace systems [5] introduced identity verification and escrow-based transactions for informal workers. While the platform strengthened transaction security, it operated as a broad marketplace without any concept of restricted community participation or localized governance.

Community-restricted digital platforms [6] addressed trust through institutional verification by limiting access to authenticated university users. This approach improved reliability within academic environments but remained limited in terms of scalability and administrative monitoring.

Other MERN-based marketplace systems [7] developed a MERN-based marketplace with multi-layered verification mechanisms. Although the platform improved provider reliability, it did not include governance tools or analytical monitoring features for long-term service management.

Luo et al. [8] further demonstrated that user trust in online marketplaces is strongly influenced by perceived community interaction and service quality. Their findings reinforce the importance of integrating trust mechanisms with community- oriented platform design.

A. Critical Analysis

A comparative review of these systems reveals that most existing platforms solve only individual parts of the hyperlocal service problem. Some focus primarily on verification, while others emphasize scheduling, accessibility, or restricted participation. However, very few systems integrate these components into a unified framework.

Another important observation is the limited role of administrative governance in current platforms. In most cases, trust mechanisms operate only during registration or through user ratings after service completion. There is little support for continuous moderation, service monitoring, or community-level quality control.

Additionally, many existing solutions are designed for specific environments such as universities or regional marketplaces, which restricts their scalability. Limited attention has been given to systems capable of supporting multiple independent residential communities while preserving localized administration and access control.

These issues show the need for a platform that combines trust, governance, scalability, and analytical monitoring within a single community-focused system. MyColony is designed to address these challenges through an integrated approach tailored specifically for residential colonies.

3. Research Gap

A review of existing hyperlocal service platforms reveals several limitations that remain insufficiently addressed:

A. Limited Community Governance

Most existing systems rely either on automated moderation through ratings or provide no moderation mechanisms at all. Very few platforms include structured administrative control where community administrators can approve, monitor, or remove service listings based on local standards and user feedback.

B. Limited Multi-Community Scalability

Many current platforms are designed for a single environment such as a university campus or a specific regional marketplace. However, residential service platforms need to support multiple independent colonies, each operating with its own administrators, users, and service providers. This type of scalable multi-community architecture is rarely addressed in existing research.

C. Lack of Operational Analytics

Another important limitation is the absence of analytical monitoring tools. Most systems provide restricted visibility into service usage patterns, user activity, provider performance, or community engagement. Without such insights, administrators cannot effectively evaluate system performance or identify areas requiring improvement.

D. Weak Integration of Trust Mechanisms

Verification systems and user rating mechanisms are often implemented separately. In many platforms, provider verification occurs only during registration, while ratings are collected independently after service delivery. As a result, there is limited support for continuous monitoring of service quality and long-term provider reliability.

E. Proposed Contribution

MyColony is designed to address these limitations through an integrated and community-focused approach. The platform combines administrator-led moderation, role-based access control, and multi-colony support within a single system architecture.

In addition, the platform incorporates analytical features that allow administrators to monitor service activity, user engagement, and provider performance. These insights support more effective decision-making and help maintain service quality over time.

Another key contribution of the proposed system is the integration of verification and monitoring mechanisms within a unified trust framework. Instead of treating trust as a one-time verification process, MyColony supports continuous evaluation through moderation and community interaction.

By integrating governance, scalability, analytics, and trust management into a single platform, MyColony provides a structured solution specifically designed for residential service ecosystems.

4. System Architecture

The MyColony platform is designed using a three-tier architecture model, ensuring scalability, modularity, and efficient data handling. The architecture is divided into three primary layers: the Presentation Layer, the Application Layer, and the Data Layer. This section details the technical framework, database structures, and operational workflows that underpin the system's functionality.

A. Three-Tier Architecture Overview

- **Presentation Layer:** Developed using React.js, this layer provides a dynamic and responsive user interface. It enables users to interact with the system — performing actions such as registration, service posting, and browsing available listings — with seamless transitions and real- time updates achieved through component-based rendering and React state management.
- **Application Layer:** Implemented using Node.js and Express.js, this layer handles all backend operations including business logic, authentication, API routing, and service processing. Communication between the frontend and backend is achieved through RESTful APIs over HTTP/HTTPS protocols, with JSON as the data interchange format.
- **Data Layer:** Consists of MongoDB, a NoSQL document- oriented database used to store user data, colony details, and service listings. MongoDB's schema-less design allows for rapid iteration on evolving data structures. Cloudinary is integrated as a cloud-based media storage service, providing optimised image storage and CDN- accelerated delivery without burdening the application server.

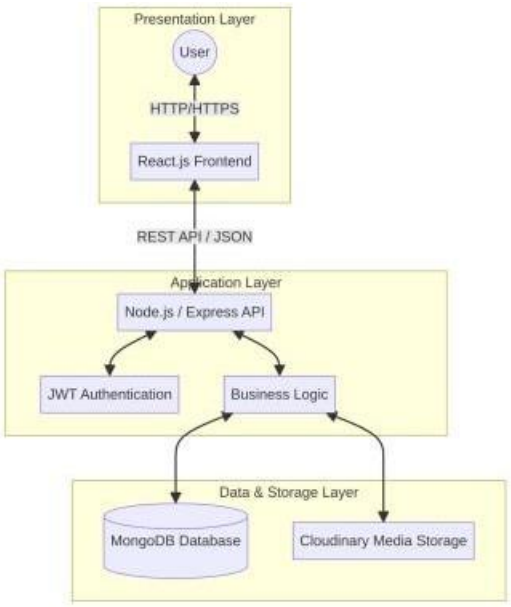

Fig. 1. Three-Tier System Architecture of MyColony (Presentation → Application → Data Layer).

The three-tier architecture was selected because it separates user interaction, business logic, and data management into independent layers. This separation simplifies maintenance and allows future modules to be integrated without affecting the entire system. During implementation, the modular design also enabled parallel development across frontend and backend components.

B. Database Design (ER Model)

The MyColony system utilises a NoSQL document-based database structured into three primary collections: Users, Colonies, and Services. The design ensures data consistency and efficient colony-scoped querying while maintaining the flexibility of non-relational data.

1) Entities and Attributes

The three core database collections, their attributes, and descriptions are defined in Table I below:

Table I – Database Collections And Attributes

Collection	Attributes	Description
Users	userID (PK), name, email, password, role, colonyID (FK)	Stores authentication credentials and colony association. Roles: user, admin.
Colonies	colonyID (PK), colonyName, location, amenities	Contains geographical and identification data for each community.
Services	serviceID (PK), userID (FK), title, description, category, images, status	Stores service details. status field (Pending/Approved) drives moderation.

2) Relationships and Data Flow

The database is designed to ensure that each User is associated with exactly one Colony via a colonyID foreign key, enabling localised service scoping. Services are linked to their owner through userID, maintaining ownership and traceability. A many-to-one relationship exists between Services and Colonies, ensuring that only colony-resident users can view listings within their community. The ER diagram below illustrates these associations:

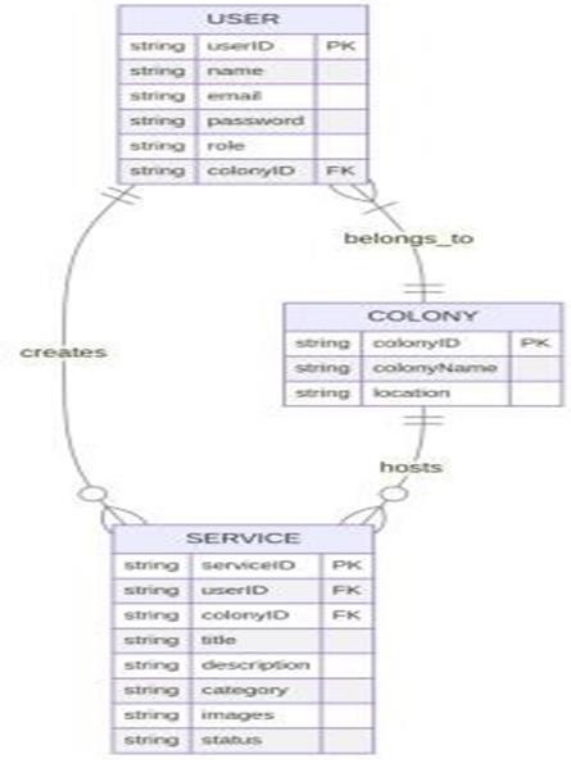

Fig. 2. Entity-Relationship (ER) Diagram: User ↔ Colony ↔ Service associations with PK/FK notation.

3) System Workflow and Process Flow

The workflow of the MyColony platform ensures smooth interaction between users and administrators while maintaining trust and quality control. The moderation mechanism is central to the platform: when a service is submitted, it is not immediately visible to other users. Instead, it undergoes a structured administrator approval process comprising four stages:

1. **Onboarding:** The user registers on the platform and joins or creates a colony via the Colony Management Module. The system assigns the user a colonyID and role (user or admin).

2. **Submission**: The user submits a service listing through a structured form. The system validates all fields and persists the document to MongoDB with status = "Pending", making it invisible to other residents.

3. **Moderation:** The colony Admin reviews the pending listing in the Admin Moderation Portal, verifying authenticity and alignment with community standards. The admin may approve or reject the listing.

4. **Activation:** Upon approval, the status field transitions to "Approved" and the service becomes visible in the colony marketplace. Rejected services are permanently removed. The entire transition is logged for audit traceability.

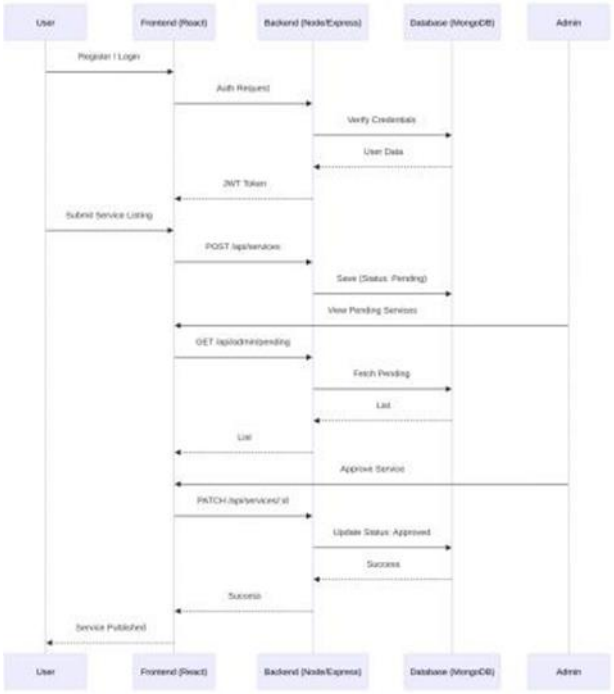

Fig. 3. End-to-End Process Flow and Sequence Diagram (User → Frontend → Backend → DB → Admin).

4) Data Flow Explanation

Data in MyColony flows through three distinct phases, ensuring consistency across users, services, and colonies at every stage of interaction:

1. **Input Phase:** The user interacts with the React.js frontend. Form submissions, search queries, and authentication requests are captured as user events and serialised into structured JSON payloads. React's component-level state management ensures real-time UI updates without full page reloads, providing a smooth user experience across devices.

2. **Processing Phase**: JSON payloads are transmitted to the Node.js/Express.js backend via RESTful API calls using standard HTTP methods (GET, POST, PATCH, DELETE). The backend validates and sanitises all input, applies JWT-based authentication middleware to verify session tokens, executes business logic (including status transitions for service moderation), and constructs optimised database queries via the Mongoose ODM layer.

3. **Persistence Phase:** Validated data is written to MongoDB collections. Colony-scoped queries ensure that all data retrieval is filtered by colonyID, preventing cross-colony data leakage and maintaining strict community isolation. Image binaries are offloaded to Cloudinary via secure API upload; only the resulting CDN URL is stored in the Services collection, eliminating server-side binary storage and reducing MongoDB document payload size. All reads and writes return JSON responses that propagate back through the API layer to update frontend state in real time.This three-phase flow helps maintain a consistent and secure path for user interactions across the platform, from registration to service approval. The stateless nature of JWT authentication allows each API request to be verified independently without maintaining server-side session data, which supports scalable application deployment.

5. Methodology

A. System Development Approach

MyColony is a full-stack, browser-based service marketplace built for residential colonies. It addresses a practical gap: the absence of a trusted, community-scoped platform through which residents can offer and discover local services such as tiffin delivery, tutoring, laundry, and home maintenance. The development follows a layered, modular architecture strategy in which each component is independently maintainable yet cohesively integrated with the rest of the system.

The technical foundation is the MERN stack — MongoDB 8.8.1, Express.js 4.21.1, React.js 18.3.1, and Node.js 22.12.0 [10]. Building every tier in JavaScript eliminates format conversion between layers and allows data structures to flow consistently from the database through the API to the browser. Node.js contributes an asynchronous, non-blocking request- handling model that enables the backend to serve multiple concurrent users without spawning a separate thread per connection [11].

The presentation layer is built with React.js 18.3.1 using a component-driven architecture. Application state is managed centrally through Redux Toolkit 2.3.0, keeping authenticated user data and colony associations consistent across all components without redundant API calls. Client-side routing is handled by React Router DOM 6.28.0. The interface is styled using Tailwind CSS 3.4.15 with a mobile-first approach, supplemented by Radix UI for accessible component primitives and Framer Motion 11.11.17 for animations. HTTP communication is managed through Axios 1.7.7.

The backend is powered by Express.js 4.21.1 on Node.js 22.12.0, which exposes a RESTful API [2] through standard HTTP verbs: GET, POST, PUT, and DELETE. File uploads are processed by Multer 1.4.5. Data persistence is handled through MongoDB 8.8.1 via the Mongoose 8.8.1 ODM, providing schema-level validation and structured relationships between collections. Field-level indexing on colony reference IDs and service approval status keeps query performance stable as collection sizes grow.

B. Technology Justification

MongoDB was chosen over relational alternatives such as PostgreSQL because the service data model in MyColony is inherently variable. For example, a tiffin service listing carries menu items and daily pricing, whereas a tutoring service

carries subject specializations and hourly rates. MongoDB's document model handles such heterogeneous records naturally within the same collection, and Mongoose's schema validation enforces the minimum required structure without restricting field variation across service categories [13].

For password security, bcryptjs 2.4.3 implements the bcrypt adaptive hashing algorithm with a salt round count of 10, making offline brute-force attacks computationally prohibitive[15]. JWT-based authentication was adopted over server-side session management because it removes the need for a session store. Each token encodes the user's ID and role, is cryptographically signed, and carries a 24-hour expiration. The server validates incoming requests by verifying the token signature rather than querying a sessions table, lowering per- request overhead and enabling horizontal scaling without shared state.

C. System Modules

Authentication Module

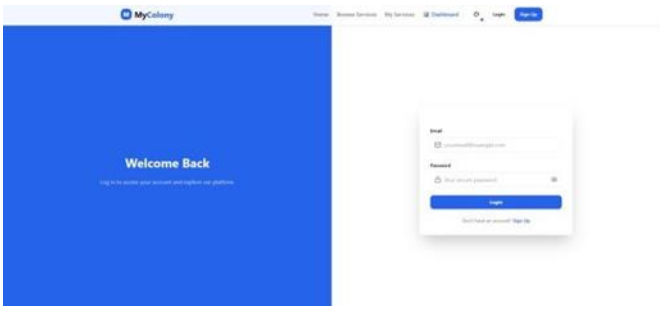

Fig. 6. Login page(Sign in page).

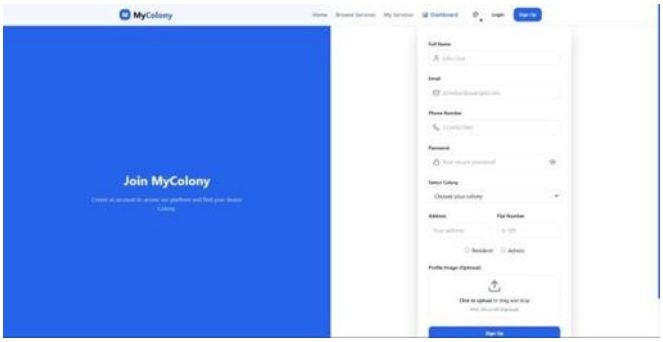

Fig. 7. Sign up page.

Submitted passwords are hashed using bcryptjs 2.4.3 at 10 salt rounds before being written to the Users collection. Upon successful login, the backend issues a JWT with a 24-hour expiry, which is stored in localStorage on the frontend and in an HTTP cookie on the backend. A middleware function intercepts every protected route, decodes the token, and attaches the verified user identity to the request object before passing control to the route handler.-

Colony Management Module

Colony creation is restricted to administrators. Each colony document stores the name, address, city, pincode, total number of flats, amenities list, and the administering user's ObjectId. During registration, residents select their colony, and its ObjectId is stored on their user document. This ObjectId acts as the primary filter when retrieving service listings, creating the hyperlocal visibility boundary that defines the platform.

Service Management Module

Residents create listings by submitting a title, description, category, pricing, availability schedule, and up to five images. Image files are processed by Multer 1.4.5 and forwarded to Cloudinary 2.5.1, which returns CDN-backed URLs stored in the Services collection. Newly created listings are flagged as unapproved and placed in a pending queue visible on the admin dashboard. Only listings that receive administrator approval are published to the colony's public service feed.

D. Media Handling And Cloud Storage

Rather than storing uploaded files on the application server, all media is delegated to Cloudinary 2.5.1. The upload pipeline works as follows: Axios packages image files into a multipart request; Multer 1.4.5 receives and buffers the files on the backend; the Cloudinary SDK then uploads each file and returns a permanent, CDN-backed URL; and these URLs are stored in the images array of the corresponding Services document. When a resident views a listing, images are loaded directly from Cloudinary's CDN, bypassing the application server entirely. This approach eliminates backend storage overhead while ensuring fast, geographically distributed image delivery.

E. Performance Metrics And Optimization

Node.js 22.12.0 uses an event-driven model that allows Express.js to handle multiple simultaneous requests without thread-per-connection overhead [11]. Under typical operating conditions, API response times remain below two seconds. Mongoose is configured with field-level indices on the colony reference field and the email field, reducing the most frequent query pattern — fetching approved services by colony — from a full collection scan to a logarithmic index lookup. On the frontend, Redux Toolkit 2.3.0 caches fetched data in the centralized store, preventing redundant API calls when residents navigate between previously loaded pages. Image assets are served through Cloudinary's CDN from geographically proximate edge nodes, keeping media load times low regardless of backend load.

F. Access Control And Security Mechanisms

Role-Based Access Control is enforced through JWT payload inspection combined with middleware-level route guards [14]. The token payload encodes each user's role — admin, resident, or user. Routes that perform administrative operations include middleware that reads the decoded role and returns a 403 Forbidden response if the role does not

meet the required permission level. Because this check occurs server- side on every request, frontend role restrictions cannot be bypassed by manipulating client-side state.

All sensitive configuration values — the MongoDB Atlas connection string, JWT signing secret, and Cloudinary API credentials — are stored as environment variables and never committed to the Git repository. CORS is configured to accept requests only from the Netlify-hosted frontend origin, blocking unauthorized cross-origin access. All client input is validated and sanitized before processing, and error responses return generic messages that do not expose internal stack traces or database schema details.

G. Testing And Validation

Testing is conducted across four complementary dimensions, each targeting a distinct category of potential failure.

Functional Testing: All primary user workflows are exercised end-to-end across all three roles, covering registration, colony selection, service listing with image uploads, admin approval and rejection, profile editing, and category-based filtering. Valid inputs are tested to confirm correct behavior, and invalid inputs are tested to confirm graceful rejection.

API Testing: All backend endpoints are verified using Postman for correct HTTP status codes (200, 400, 401, 403, and 404), response body structure, and JWT middleware behavior when tokens are absent, malformed, or expired. Cloudinary upload endpoints are tested with both valid images and unsupported file types to verify that Multer rejects invalid uploads before they reach the cloud storage layer.

Load Testing: Multiple simultaneous API calls are simulated against the service listing retrieval and authentication endpoints to confirm that Node.js's non-blocking architecture sustains sub-two-second response times under concurrent usage [11] without request failures.

Security Testing: Requests without tokens, with expired tokens, and with incorrectly signed tokens are each verified to return 401 Unauthorized. Role escalation attempts — such as a resident-role token targeting an admin-only endpoint — are verified to return 403 Forbidden. Input fields are tested with injection strings and oversized payloads to confirm that validation middleware rejects them before data reaches the database

H. Deployment Strategy

The frontend React application is compiled using Vite 6.0.3 and deployed to Netlify, which distributes the compiled static assets across a global CDN. The backend Express.js API is deployed on Render, which is connected to the project's GitHub repository and triggered to redeploy automatically whenever changes are merged to the main branch. The database is hosted on MongoDB Atlas with automated daily backups, connection pooling, and IP allowlist-based access control [10]. All image assets are served through Cloudinary's CDN. This four-platform deployment architecture ensures that no single tier becomes a bottleneck, as each layer scales independently according to its own demand profile.

I. Limitations

Despite its strengths, the current implementation has several acknowledged constraints. First, issued JWT tokens cannot be invalidated before their 24-hour expiry without introducing a server-side token blacklist [14], meaning that a compromised token remains usable until it naturally expires. Second, load testing was conducted in a controlled local environment; real- world performance under large concurrent user bases across geographically distributed colonies may differ and warrants further evaluation. Third, the service approval workflow depends entirely on manual administrator review, which may become a bottleneck as the volume of submitted listings grows. Fourth, MongoDB's flexible schema requires strict application-level validation to compensate for the absence of database-enforced constraints [13] any gap in that validation layer risks allowing malformed documents into the collections. These limitations are acknowledged as priorities for future architectural refinement.

J. Methodology Summary

The methodology underlying MyColony reflects deliberate technical decisions oriented around three goals: community trust, long-term maintainability, and infrastructure scalability. The MERN stack com[10], bcrypt password hashing [15] at 10 salt rounds, JWT authentication [14] with a 24-hour expiry, Cloudinary-backed media storage, MongoDB Atlas hosting, and a Netlify-plus-Render deployment pipeline together form a coherent technical foundation in which every

component serves a defined purpose. The modular three-tier structure makes future feature integration easier — such as real-time chat, payment gateway integration, or a React Native mobile application — to be incorporated without disrupting the existing architecture. Testing across functional, API, load, and security dimensions confirms that these design decisions translate into reliable, production-grade system behavior.

6. Results And Discussion

This section explains the outcomes observed after the development and testing of the MyColony platform. It discusses how the system behaved during implementation, how users interacted with it, and how well it addressed the intended purpose of supporting service management within residential communities. The discussion also considers usability, administrative control, system responsiveness, and the broader value of the platform compared with existing alternatives. Based on the observations gathered during testing, the platform demonstrates that a localized digital service system can improve convenience, organization, and trust among residents while remaining practical for future expansion[16].

A. Implementation Outcomes and Interface Analysis

The MyColony platform was implemented as a web-based application using a modern software stack that allows different parts of the system to communicate efficiently[17]. During implementation, particular attention was given to designing an interface that is straightforward and easy to navigate. The resulting layout enables users to interact with the platform with minimal effort, even if they do not have advanced technical knowledge. Menus, forms, and service displays were organized clearly so that users could access information without confusion.

User Authentication and Onboarding

One of the most significant outcomes of the implementation phase is the successful integration of a secure authentication mechanism[18]. Access to the platform is limited to registered users, ensuring that only verified individuals can participate in the system. This restricted-entry design supports privacy and contributes to a more trustworthy digital environment. In a residential setting where users may be interacting with neighbors or service providers within a shared locality, the presence of secure login controls increases confidence in the legitimacy of the community.

Service Discovery and Marketplace

The main user dashboard functions as a centralized area where available services can be viewed and explored. Instead of depending on fragmented channels such as personal messages, social media posts, or informal group chats, users are provided with a single platform containing structured service information. This arrangement improves accessibility because residents can browse listings in an organized manner, compare available options, and identify suitable services more quickly[19]. It also reduces the inefficiency often caused by searching through scattered announcements.

Administrative Moderation Portal

Another important result of implementation is the development of an administrative panel that allows moderators or administrators to review service submissions before they are published. This feature plays a central role in preserving the quality of the platform. By checking submitted listings for relevance and appropriateness, administrators help prevent misleading, incomplete, or unsuitable services from appearing to other users. This moderation process contributes directly to the reliability of the system and distinguishes the platform from informal community communication tools[20].

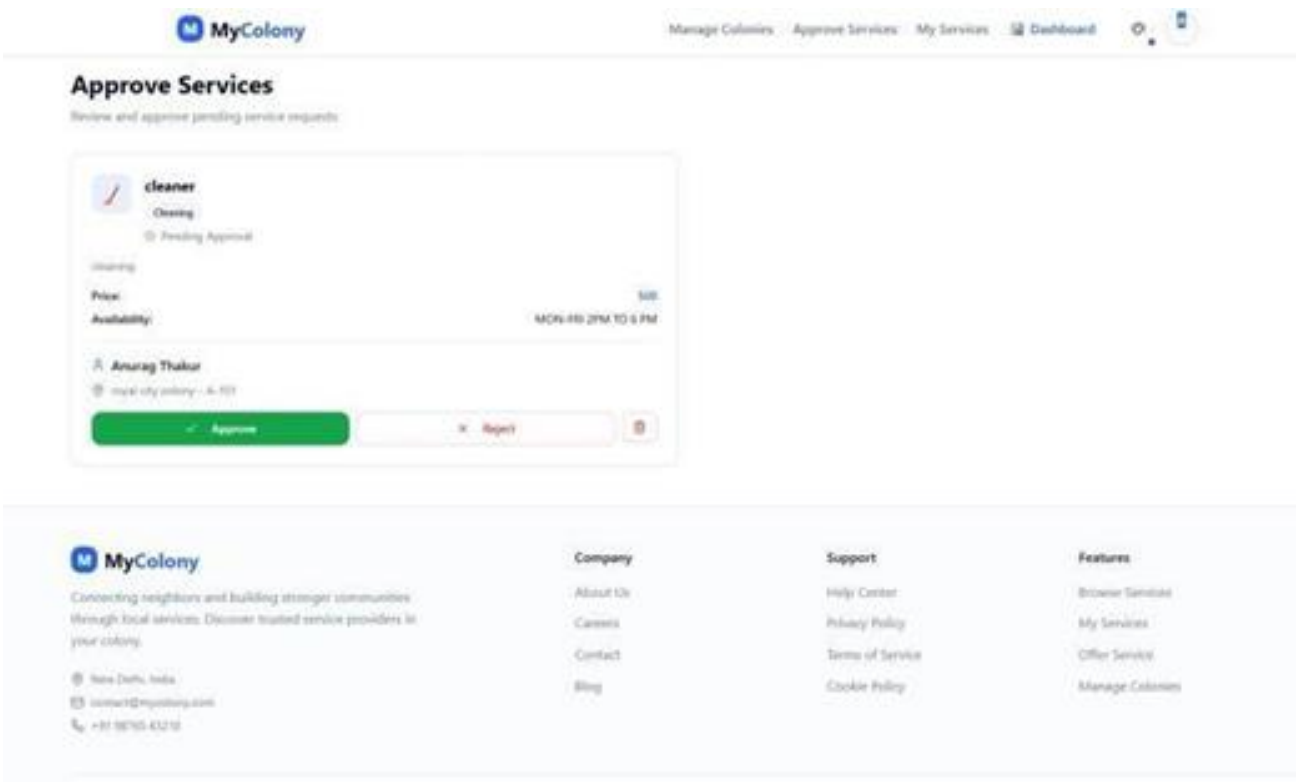

Fig. 8. Administrative Moderation Portal (Approve Services).

Service Contribution Interface

The platform also supports community participation by enabling users to submit their own service offerings. The listing form was designed to collect essential information in a guided and structured way. Users are prompted to enter details such as service category, price, description, and availability. This not only improves the quality of submitted content but also encourages users to contribute because the process is simple and understandable. As a result, the platform supports both service discovery and service contribution, creating a more interactive local exchange environment.

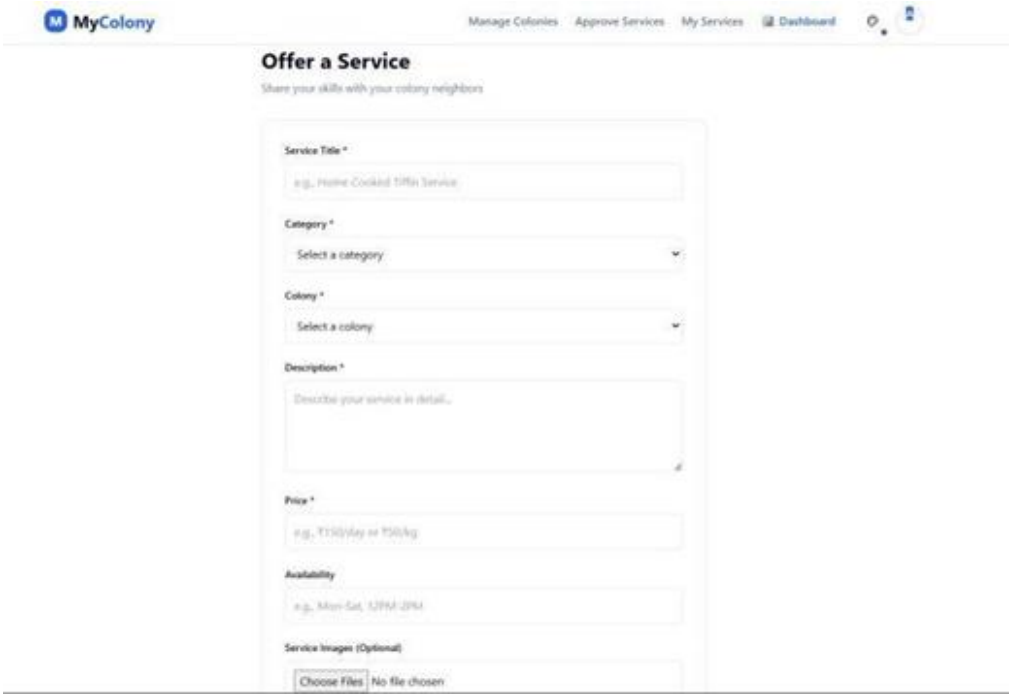

Fig. 9. Offer a Service Interface.

Colony Management

A further implementation outcome is the system's support for multiple residential groups. Services can be organized according to specific communities, ensuring that content remains locally relevant. This separation is especially valuable in cases where different residential areas have unique needs, preferences, or service providers. By maintaining this localized structure, the platform avoids becoming overly generalized while still allowing future scaling and broader adoption.

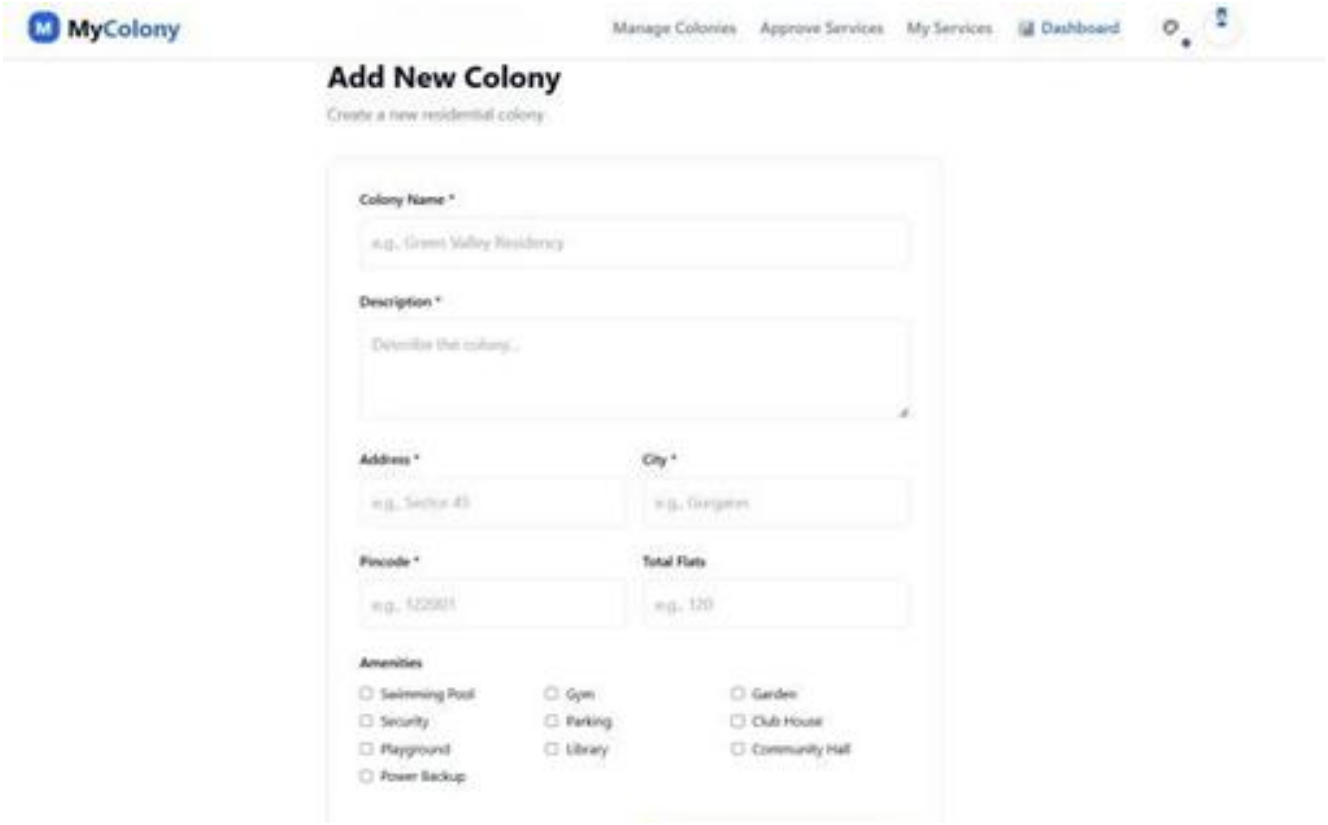

Fig. 10. Colony Management Interface (Add New Colony).

B. Functional Validation and Data Analysis

The functional performance of MyColony was assessed by observing user interactions and analyzing how the system handled service-related activity during testing. Several indicators were considered, including the number and type of submitted services, approval outcomes, browsing behavior, and participation patterns across different residential groups[21]. These observations provide insight into how effectively the platform fulfills its intended role.

Service Category Distribution

The distribution of service categories indicates that some types of services are more commonly offered and accessed than others. In particular, educational assistance and household-related services were among the most frequently observed categories. This suggests that these areas represent practical and recurring needs within residential communities. Other service types, including food services and cleaning- related tasks, were also present, demonstrating that the platform can accommodate a range of everyday requirements. The variety of categories highlights the flexibility of the system and its relevance to community- based living.

Table 2 Service Category Distributio

Category	Demand	Description
Tutoring	High	Educational support for students.
Maintenance	High	Plumbing, electrical, home repairs.
Laundry/Cleaning	Medium	Household chores and garment care.
Food/Tiffin	Medium	Home-cooked meals and catering.

Fig. 11. Graph 1 – Service Category Distribution.

Administrative Approval Workflow

The approval workflow was also evaluated to determine whether the moderation mechanism functioned effectively. Most of the submitted services were approved after administrative review, indicating that users generally understood how to provide acceptable and useful listings. This reflects positively on both the submission interface and the platform guidelines. A high approval rate suggests that the system successfully encourages meaningful contributions rather than low-quality or irrelevant content. At the same time, the existence of a review stage ensures that standards are maintained before services become publicly visible.

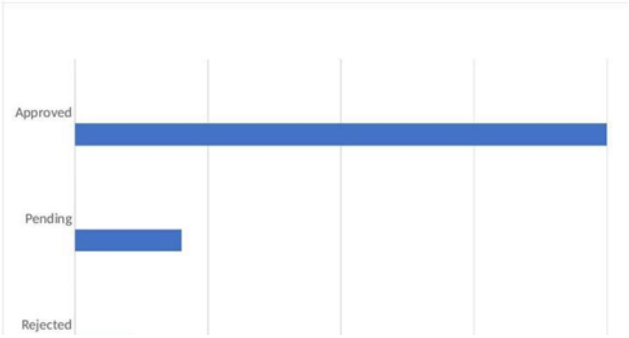

Fig. 12. Graph 2 – Service Approval Status.

User Activity Metrics

User engagement data further confirms the usefulness of the platform. During testing, the number of service views exceeded the number of service submissions, which indicates that browsing activity is strong even among users who are not actively posting their own listings. This pattern suggests that the platform serves an important informational role.

Many residents may primarily use the system to search for available services rather than offer them, and the platform still provides value for these users by making relevant services easy to find.

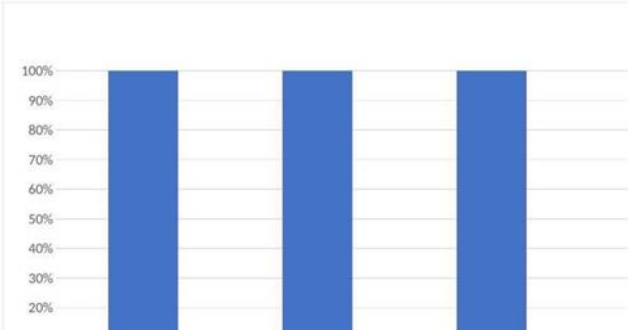

Fig. 13. Graph 3 – User Activity Analysis.

Colony-wise User Engagement

Participation patterns were not identical across all residential groups. Some communities showed relatively high levels of activity, while others were less engaged. This difference may be influenced by factors such as community size, awareness of the platform, familiarity with digital tools, or the level of demand for local services. These variations suggest that technical readiness alone is not enough to ensure widespread use; user adoption strategies, orientation, and promotion are also important for the long-term success of the system.

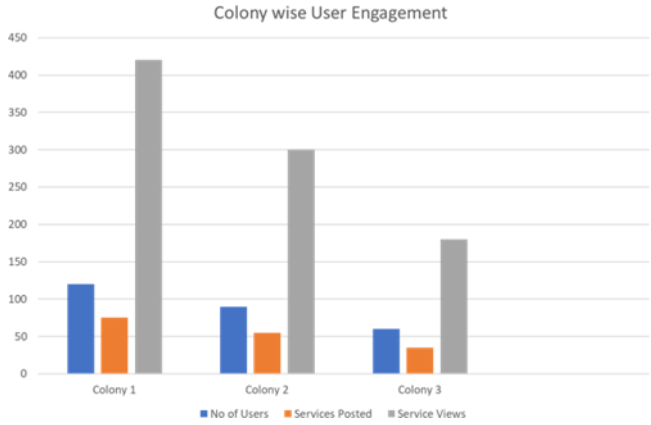

Fig. 14 Colony wise User Engagement

Analysis of System Performance

The platform displayed stable and dependable behavior throughout the testing period. No major technical issues were observed that significantly affected accessibility or usability. This stable operation is an important result because it shows that the system can support routine service browsing, submission, and moderation without disrupting the user experience.

The presence of consistently used service categories suggests that the platform aligns well with real community needs.

Users appear to be engaging with the types of services that are most relevant to daily life within residential areas, which strengthens the practical value of the system. A platform may be technically functional, but its effectiveness ultimately depends on whether it supports meaningful interactions. In this case, the observed usage patterns indicate that the system is not only operational but also useful.

The moderation process also contributes substantially to overall performance. By requiring each listing to be reviewed before publication, the system reduces the possibility of inaccurate, misleading, or inappropriate content reaching users.

This process strengthens trust in the platform and improves the consistency of the available information. For a localized service platform, trust is especially important because residents may base real decisions on the listings they encounter.

Another notable finding is that the system is capable of supporting multiple residential groups without a visible decline in responsiveness. This suggests that the overall architecture is suitable for handling growth. As more communities or users are added, the platform has the potential to remain functional and manageable, provided that future optimization and resource planning are carried out appropriately.

Comparison with Existing Systems

When compared with existing service-sharing approaches, MyColony offers several meaningful advantages. Large public marketplaces typically operate at a broad scale and are designed to reach wide audiences. Although these platforms provide convenience, they often do not prioritize the specific needs of a single residential community. As a result, users may encounter listings that are too broad, too distant, or insufficiently tailored to local conditions.

Informal systems such as neighborhood group chats or messaging applications offer immediacy and familiarity, but they often lack structure, verification, and record management. Service posts may quickly become buried in conversation threads, and there may be no clear process for reviewing information before it is shared. This can create uncertainty about accuracy and reduce user confidence.

MyColony provides a middle-ground solution by combining localized relevance with platform-based organization.

Services are displayed in a structured format, user access is controlled, and administrative review adds an additional layer of quality assurance. This combination makes the platform more dependable than informal communication channels [4] while still preserving the community focus often missing from large online marketplaces.

In addition, the platform can generate insights about user behavior and service demand. These observations can help administrators identify which categories are most needed, which groups are most active, and where improvements may be required. This analytical value is not commonly available in purely informal systems and represents another practical strength of the platform.

System Performance Evaluation

From a technical standpoint, the platform performs efficiently under normal testing conditions. The backend infrastructure is able to process multiple requests in a timely manner, allowing users to browse, submit, and review services without noticeable delays. This responsiveness is essential for maintaining a smooth user experience.

Communication between the application layers was observed to be consistent and reliable. Data transfer between the front- end interface, server-side processing, and storage components occurred without significant disruption. The system responded within acceptable time limits during testing, indicating that the architecture is suitable for the intended scale of operation.

The user interface also contributes to overall performance by allowing actions to be completed dynamically. Rather than forcing the user to reload entire pages repeatedly, the system supports faster interaction and more convenient navigation. This improves usability, particularly for users who want to compare several services in a short time.

Media handling was further optimized through the use of external storage solutions for files such as images. This reduces the burden on the core application server and improves the speed at which visual content is loaded. As a result, the system maintains better efficiency while still supporting richer service listings.

Limitations:

Despite the positive results, several limitations remain. One of the main constraints is the lack of an integrated payment system. Because transactions must be completed outside the platform, users may experience reduced convenience and less transparency during the payment process.

Another limitation is the absence of built-in communication tools such as direct messaging. Users must rely on external channels to contact service providers, which may slow interactions and reduce the seamlessness of the overall

experience. These limitations do not prevent the platform from functioning, but they highlight areas where the service process can be improved.

Future Work:

There are several opportunities to strengthen the platform in future versions. Integrating an internal payment feature would make transactions more convenient and allow users to complete the full service process within a single environment. Adding personalized recommendations could also improve usability by helping residents discover services that match their preferences or previous activity.

A mobile application or mobile-optimized version would further increase accessibility, especially for users who rely primarily on smartphones. Additional communication features, including messaging and notifications, could make the platform more interactive and efficient by improving coordination between users and service providers.

Discussion: Addressing the Research Gap

The platform contributes to addressing the gap in community-based digital service management by offering a structured and monitored environment rather than an open and loosely organized exchange space. Its design demonstrates that local service systems can be made more dependable when verification, moderation, and organized presentation are built into the platform from the start.

Administrative involvement and usage data together help improve service quality, user confidence, and platform oversight. In this way, MyColony responds to common weaknesses found in existing solutions, particularly those related to trust, inconsistency, and lack of localized structure.

Table 3 Platform Feature Comparison

Feature	**MyColony (Proposed)**	**Generic Marketplaces**	**Informal Channels**
Trust Layer	**Colony-Level Admin**	Broad/Global	Personal Referrals
Verification	**Role- Based/Colony-Specific**	General Identity	None
Governance	**Local Admin Moderation**	Platform-wide	None
Analytics	**For Colony Managers**	For Providers	None

Performance Evaluation

The performance of the MyColony platform was examined by focusing on important aspects such as server-side processing, interface responsiveness, and efficient use of system resources. The evaluation indicates that the platform performs reliably and is suitable for practical deployment in real-world environments.

Backend Efficiency:

The server-side implementation, built using Node.js, follows an asynchronous and event-driven approach that allows it to manage multiple user requests at the same time without slowing down the system. This design helps maintain consistent performance even when the number of active users increases. In addition, the use of REST-based APIs with lightweight JSON data exchange ensures quick communication between the client and server, reducing delays and providing stable response times.

UI Fluidity:

The frontend of the platform is developed using React.js, which improves the overall user experience by updating only the necessary parts of the interface instead of reloading entire pages. This approach enables faster interactions and smoother navigation. Users can browse services, submit information, and interact with the system without noticeable interruptions, resulting in a more responsive and user- friendly interface.

Resource Optimization:

To manage media efficiently, the platform uses Cloudinary, which stores and delivers images through a distributed network. This reduces the burden on the main server and improves the speed at which visual content is loaded.

Additionally, the use of token-based authentication removes the need to store session data on the server, allowing the system to scale more easily and use resources more effectively.

Overall Evaluation:

Considering all these factors, the MyColony platform demonstrates strong performance in terms of stability, speed, and scalability. It is capable of supporting increasing user activity while maintaining consistent responsiveness. These characteristics make it well-suited for continued growth and real-world usage.

7. Conclusion

The results of implementation and testing show that MyColony is an effective platform for managing services within residential communities. It improves access to local services through a structured, moderated, and user-friendly digital environment. User activity patterns indicate that the system is relevant to community needs, while the technical evaluation confirms that it performs reliably and can support future growth.

Overall, the platform presents a practical and scalable approach to local service management. Although some features remain to be added, the current system already offers clear value by improving organization, accessibility, and trust within residential service exchange.

8.References

[1] Luo, N., Wang, Y., & Zhang, M. (2020). Integrating community and e-commerce to build a trusted online second-hand platform. Technological Forecasting and Social Change, 153, Article 119913.
[2] Resnick, P., Zeckhauser, R., Friedman, E., & Kuwabara, K. (2002). Trust among strangers in internet transactions: Empirical analysis of eBay's reputation system. Advances in Applied Microeconomics, 11, 127–157.
[3] Sundararajan, A. (2016). The sharing economy: The end of employment and the rise of crowd-based capitalism. MIT Press.
[4] Rayle, S., Dai, D., Chan, N., Cervero, R., & Shaheen, S. (2016). Just a better taxi? A survey-based comparison of taxis, transit, and ridesourcing services. Transport Policy, 45, 168–178.
[5] Botsman, R., & Rogers, R. (2010). What's mine is yours: The rise of collaborative consumption. Harper Business.
[6] Davidson, S., & Infranca, M. (2016). The sharing economy as an urban phenomenon. Yale Law & Policy Review, 34(2), 215–279.
[7] Rosenblat, M., & Stark, L. (2016). Algorithmic labor and information asymmetries: A case study of Uber's drivers. International Journal of Communication, 10, 3758–3784.
[8] Hamari, J., Sjöklint, M., & Ukkonen, A. (2016). The sharing economy: Why people participate in collaborative consumption. Journal of the Association for Information Science and Technology, 67(9), 2047–2059.
[9] Future Market Insights. (2025). Hyperlocal services market forecast and outlook 2025–2035.
[10] MongoDB. (2024). MongoDB documentation. Retrieved from https://www.mongodb.com/docs/
[11] Node.js. (2024). Node.js official documentation. Retrieved from https://nodejs.org/en/docs/
[12] React. (2024). React official documentation. Retrieved from https://react.dev/
[13] Web Development. (2022). Understanding the MERN stack: A comprehensive guide. International Journal of Computer Science Trends and Technology.
[14] JSON Web Token. (2021). JSON Web Token (JWT) authentication and authorization in modern web applications. IEEE Access.
[15] Provos, N., & Mazieres, D. (1999). A future-adaptable password scheme. USENIX Annual Technical Conference.
[16] Satija, H. (2026). MyColony: A hyperlocal service management platform for residential communities (Project report).
[17] OpenJS Foundation, & Meta. (2025). Node.js documentation and React.js documentation.
[18] Hardt, D. (2012). The OAuth 2.0 authorization framework (RFC 6749). IETF.
[19] Linden, G., et al. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet Computing.
[20] Resnick, K., et al. (2000). Reputation systems for online communities. Communications of the ACM.
[21] Nielsen, J. (1994). Usability engineering principles for user interaction.