Pujanam: A Comprehensive Portal For Pandit Booking, Puja Services & Samagri Management

Bhushan Gavhane[1], Soham Utpat[2], Vishal Jadhav[3], Manohar Chaudhari[4]
[1, 2, 3, 4] Dept of Computer Engineering
[1, 2, 3, 4] Sinhgad Institute of Technology Pune, India

Abstract- *This paper introduces Pujanam, a web-based platform that consolidates pandit booking, puja scheduling, and samagri procurement into a single digital workflow. Arranging religious services through conventional means—phone calls, word-of-mouth referrals, and in-person visits—often leads to scheduling conflicts, price uncertainty, and limited reach for both devotees and pandits. Pujanam targets this operational gap through a role-differentiated system serving devotees, pandits, and administrators.*

Devotees can browse a categorized puja catalog, inspect pandit profiles complete with ratings and experience summaries, select available time slots, and place samagri orders. Pandits access a scheduling dashboard, receive live booking alerts via Socket.IO, and maintain a transaction history. Administrators oversee the full platform through a dedicated control panel covering user management, service listings, and support resolution.

The backend is built on Node.js/Express with JWT authen-tication, bcrypt password hashing, and MongoDB for schema-flexible storage. Socket.IO drives the event-based notification layer. Together, these components deliver a cohesive system that lowers coordination friction and broadens access to religious services for urban communities.

Keywords: Pandit Booking System, MERN Stack, Web Application, REST API, Real-Time Systems, Socket.IO, MongoDB, Digital Services

I. INTRODUCTION

Locating a qualified pandit for a home puja in Indian cities still relies heavily on informal networks—a relative's contact, a temple noticeboard, or a neighbor's referral. This approach works within tight-knit communities but falters for urban residents who may have relocated recently, lack established local contacts, or need to book on short notice. Even when a pandit is found, there is no structured channel to verify credentials, compare service rates, or lock in a time slot. The samagri problem runs in parallel: devotees without deep familiarity with a given ritual often overpurchase at local shops or miss required items entirely.

Pujanam—named after the Sanskrit term for worship—is a MERN-stack web application designed around these practical friction points. The system partitions functionality across three user roles: devotees who initiate bookings, pandits who accept and fulfil them, and administrators who govern platform integrity.

A typical devotee session begins with service selection from an occasion-based catalog, proceeds to a filtered pandit search, and concludes with a booking form submission that pushes an immediate real-time notification to the chosen pandit via Socket.IO. Pandits interact with a dashboard that surfaces today's schedule, pending requests awaiting response, and cumulative earnings data. Administrators hold platform-wide authority: they can approve or deactivate pandit accounts, resolve flagged support tickets, and review aggregate booking analytics.

The technology choices reflect the demands of a notification-heavy, multi-user application. React.js handles the component-driven frontend; Node.js with Express manages API routing and business logic; MongoDB stores heterogeneous service and booking documents without schema rigidity; Socket.IO sustains persistent connections for real-time updates; and JWT with bcrypt secures the authentication layer. This paper proceeds as follows: Section 2 addresses inter-face design and usability; Section 3 describes the development methodology and architectural decisions; Section 4 presents system outputs; Section 5 documents challenges and limitations; Section 6 covers author contributions; and Section 7 concludes with future directions.

II. EASE OF USE

The central design constraint for *Pujanam* was inclusivity across technical ability levels. The platform must serve a first-generation smartphone user booking a Satyanarayan Puja as comfortably as it serves a pandit reviewing a day's schedule from a tablet.

A. Interface and Navigation

A persistent top navigation bar—Home, Services, Find Pandit, About, Contact—appears on every page, giving usersa reliable orientation anchor regardless of how deep they navigate. The layout uses a responsive grid that transitions from a three-column desktop arrangement to a stacked single-column view on mobile without sacrificing any functionality. Menu depth is limited to two levels throughout the application, preventing the cognitive overload that nested navigation tends to produce.

Color is applied with a functional purpose: green marks available pandits, amber denotes pending or in-progress book-ings, and red indicates cancellations or errors. This color scheme remains consistent across the devotee, pandit, and admin interfaces, so returning users do not need to relearn visual cues when switching roles.

B. The Booking Flow

The booking path was condensed to four discrete steps, each addressing one decision point:

Step 1—Select a Puja: Services are grouped by occasion (e.g., Griha Pravesh, Navgraha Puja, Birthday Puja). Each listing displays an estimated duration and price range, letting devotees compare options before committing.

Step 2—Discover a Pandit: The search module supports filtering by city, spoken language, offered service types, star rating, and years of active practice. Contact details are masked for unauthenticated visitors to incentivize account creation while still permitting uninhibited browsing.

Step 3—Submit the Booking Form: The form captures name, phone number, full address, preferred date and time, and optional ritual notes. Client-side validation identifies empty required fields and malformed phone numbers before the form is submitted, cutting out preventable server round-trips.

Step 4—Receive Confirmation: A success screen appears immediately on submission. Concurrently, a Socket.IO event fires to the target pandit's browser session, displaying a notification badge without requiring a page reload.

C. Role-Specific Dashboards

Each role has a purpose-built landing view. The devotee dashboard lists all bookings across three status tabs (upcoming, completed, cancelled) and unlocks a rating widget once a service is marked complete. The pandit dashboard leads with today's confirmed appointments, followed by a queue of pending requests, and then a performance summary (average rating, monthly booking count). The admin dashboard prior-itizes actionable items: new registrations awaiting approval, unresolved support tickets, and accounts flagged for review.

D. Accessibility Considerations

Semantic HTML is used structurally: headings convey doc-ument hierarchy, aria-label attributes describe interactive controls, and form inputs are explicitly linked to visible labels rather than relying on placeholder text as a substitute. Error messages use plain language—"Enter a 10-digit mobile num-ber" rather than "Validation error: E_PHONE_FORMAT"—so that non-technical users understand what to correct. Spinner components accompany all asynchronous operations with ac-companying status text, keeping users informed during net-work delays.

E. Real-Time Interaction via Socket.IO

Socket.IO connections are initialized on login and held for the session duration. Each pandit is assigned a private socket room keyed to their database ID; booking events are emitted only to the relevant room, preventing notification bleed across unrelated accounts. When a pandit responds to a request, the devotee's dashboard status field updates immediately through the reverse channel, eliminating the need for manual page refreshes to check booking state.

F. System Integrity

Validation operates at three layers. React state management catches basic client-side errors. Express middleware applies business-rule checks on incoming request bodies. Mongoose schemas enforce field types, required constraints, and enum values at the persistence layer, ensuring that even direct API calls (e.g., via Postman or curl) cannot insert struc-turally invalid documents. Route-level middleware chains—verifyToken followed by requireRole—guarantee that pandit endpoints remain inaccessible to devotee tokens and vice versa.

III. SYSTEM DEVELOPMENT APPROACH

A. Architectural Design

Pujanam is organized as a three-tier application. The React client communicates with the Express API server over HTTPS; the MongoDB instance is never directly reachable from the browser. This boundary makes it practical to expose

the same API to a future mobile client or third-party integration without restructuring the backend.

Within the server, an MVC organization separates concerns: route files define URL patterns and HTTP verbs, controller functions hold request-handling and business logic, and Mon-goose models declare data schemas. Cross-cutting utilities—JWT helpers, centralized error formatters, Socket.IO event emitters—are grouped in a shared /utils module.

B. Technology Stack

Table I lists the core technologies and versions used in the implementation.

TABLE I
TECHNOLOGY STACK SUMMARY

Layer	Technology	Version
Frontend	React.js	18.2.0
Backend	Node.js / Express	18.x / 4.18.2
Database	MongoDB	6.0
Real-time	Socket.IO	4.5.4
Auth	JWT	9.0.0

React's component model let the team build reusable UI units—a BookingCard renders consistently on the devotee dashboard and on the admin audit page—reducing duplication and easing maintenance. Node.js was chosen on the server side because its event-loop architecture handles many con-current Socket.IO connections without thread-per-connection overhead. MongoDB's document model accommodates puja data naturally: a Griha Pravesh service has a different samagri list than a birthday puja, and a flexible schema avoids the sparse-column problem that would arise in a strictly relational design.

C. Authentication and Access Control

Registration stores passwords as bcrypt hashes with a cost factor of 12, providing resistance to offline dictionary attacks while keeping login latency acceptable. Successful login returns a signed JWT containing the user's ID and role. This token travels with every subsequent API request in the Authorization: Bearer header and is decoded by middleware that attaches the verified identity to the request object for downstream controllers.

Access control is enforced through middleware chains on individual routes. A protected admin endpoint, for example, passes through verifyToken (confirms the token is valid and unexpired) and then requireRole('admin') (con-firms the embedded role claim) before reaching the controller. This keeps authorization decisions declarative and out of business logic.

D. API Design

Endpoints follow REST conventions with consistent URL structure and HTTP verb semantics. Table II summarizes the primary routes.

TABLE II
API ENDPOINTS SUMMARY

Method	Endpoint	Purpose	Auth
POST	/user/register	New account	None
POST	/user/login	Issue JWT	None
GET	/user/profile	Fetch profile	User
POST	/admin/login	Admin auth	None
GET	/admin/pandits	List pandits	Admin
POST	/bookings	Create booking	User
GET	/pandit/notifications	Fetch alerts	Pandit

All error responses use a uniform JSON envelope—`{success: false, message: "...", code: HTTP_STATUS}`—so the frontend can handle errors through a single Axios interceptor rather than per-route parsing logic.

E. Development Workflow

Development followed a feature-branch Git workflow: each module was built on a named branch, reviewed via pull re-quest, and merged only after local testing passed. Postman col-lections were maintained alongside the codebase as living API documentation. React DevTools profiler sessions identified unnecessary re-renders in the booking list components, which were resolved by memoizing derived state with useMemo.

IV. SYSTEM OUTPUT / RESULTS

Figures 1 through 4 illustrate the key screens delivered in the current build.

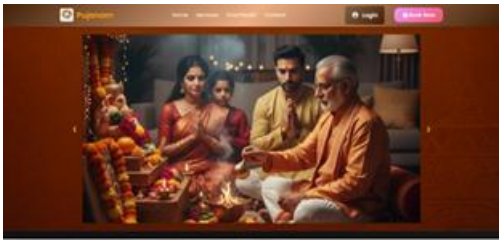

Fig. 1. Homepage of the Pujanam Platform

Fig. 2. Puja Services Catalog Page

Fig. 3. Booking Details View

Fig. 4. Administrator Control Panel

Fig. 5 shows the high-level system architecture: browser requests reach the Express API server, which interacts with MongoDB for persistence and with the Socket.IO layer for real-time event dispatch. Fig. 6 depicts the entity-relationship structure linking the Users, Pandits, Bookings, Services, and SupportTickets collections.

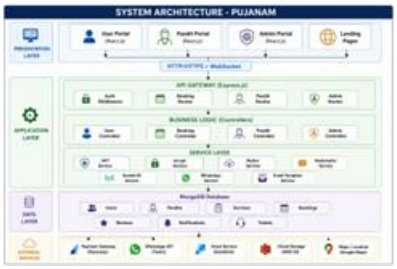

Fig. 5. System Architecture of Pujanam

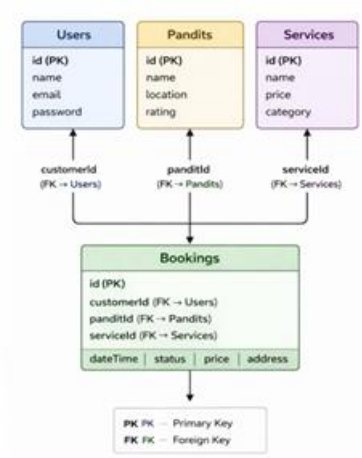

Fig. 6. Entity Relationship Diagram of Pujanam Database

V. CHALLENGES AND LIMITATIONS

A. Technical Challenges Encountered

Socket.IO's pub/sub model required deliberate room assign-ment to contain events within appropriate session scopes. An early version broadcast all booking events to a global channel, so every connected pandit received every new notification—a bug that surfaced in multi-user integration testing and was fixed by routing events to per-pandit rooms named by database ID.

Authentication state management across three roles introduced storage complexity. The team adopted separate localStorage keys per role (userToken, panditToken, adminToken), each validated by its own middleware chain. Profile photo uploads required multer middleware with explicit MIME-type filtering to reject non-image payloads before they reached the storage layer.

B. Feature Gaps in the Current Version

Payment processing is not yet integrated. Bookings are recorded in the system, but financial transactions take place offline—a significant limitation for commercial viability. Ra-zorpay and PayU have been identified as candidate gateways for a subsequent release.

Samagri ordering is partially implemented: item catalogs are viewable, but the cart and checkout flow was not completed within the project timeline. The absence of a native mobile application means the platform depends on responsive web design for phone access, which functions adequately but foregoes native push notifications and offline capability.

The platform currently supports English only. Given that a meaningful share of the target demographic—devotees in Ma-harashtra and neighboring states—navigates services

primarily in Marathi or Hindi, language coverage is a practical barrier to broad adoption.

C. Scalability Constraints

The current deployment is single-instance with no load balancer. MongoDB indexes are defined on the most-queried fields (userId, panditId, bookingDate) and list endpoints use server-side pagination to bound response sizes, but horizontal scaling has not been load-tested. The Socket.IO server would require a Redis-backed adapter (using socket.io-redis) to function correctly across multiple Node instances in a scaled deployment.

D. Security Gaps

Issued JWTs carry no server-side revocation mechanism. A token remains valid until its expiry window closes even if the associated account is suspended or the user logs out. This is a known limitation of stateless tokens; a refresh-token rotation pattern or a server-maintained token blacklist would address it. Rate limiting on authentication endpoints and two-factor authentication are absent from the current build and are earmarked for the next development phase before any public release.

E. Authors and Contributions

The *Pujanam* project was developed collaboratively by four undergraduate students in the Department of Computer Engineering at Sinhgad Institute of Technology, Pune. Work was divided along system layers: frontend development (React component architecture, routing, responsive layout), backend API development (Express route handlers, controller logic, au-thentication middleware), database schema design and index-ing strategy, and system integration with end-to-end testing. The project was conducted under the supervision of Prof. M.S. Chaudhari, who provided technical guidance and ensured the work met academic publication standards.

VI. CONCLUSION

Pujanam addresses a specific, everyday coordination prob-lem: the friction involved in finding, booking, and preparing for a home puja in an urban setting. By consolidating pandit discovery, service scheduling, and samagri listing into a single web interface—and by wiring real-time notifications through Socket.IO to eliminate the follow-up calls that currently fill that gap—the platform meaningfully reduces the effort re-quired on both sides of the transaction.

The MERN stack proved suitable for the workload: React's component architecture kept the multi-role interface main-tainable, MongoDB's document model handled service data variation cleanly, and Socket.IO's event model matched the notification patterns the booking workflow generates.

Priority additions for the next iteration include payment gateway integration to replace offline transactions, a React Native application to extend reach and enable native push alerts, multilingual support beginning with Marathi and Hindi, and security hardening through refresh-token rotation and endpoint rate limiting. With those additions, Pujanam would be production-ready for deployment across Maharashtra and could serve as a template for digitizing other culturally specific service coordination problems.

REFERENCES

[1] V. Kumar, R. Chauhan, and M. Yadav, "E-Pandit: Automated Voice-Based System for Religious Pujas," in *Proc. Int. Conf. on Reliability, Infocom Technologies and Optimization (ICRITO)*, IEEE, 2020.

[2] D. Singh and P. Sharma, "Online Temple Management System Using Web Technologies," *Int. Journal of Emerging Technologies and Innova-tive Research (JETIR)*, vol. 6, no. 4, pp. 450–456, 2019.

[3] M. Patel and R. Mehta, "Web-Based Platform for Puja Booking and Samagri Delivery," *Int. Journal of Computer Applications*, vol. 182, no. 40, pp. 25–30, 2021.

[4] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 8th ed. New York: McGraw-Hill, 2014.

[5] ReactJS Documentation. [Online]. Available: https://react.dev/

[6] Node.js Documentation. [Online]. Available: https://nodejs.org/

[7] MongoDB Documentation. [Online]. Available: https://www.mongodb.com/docs/

[8] Socket.IO Documentation. [Online]. Available: https://socket.io/docs/