yoUR Feedback Hub

A Web-Based Feedback and Grievance Management System

1. Abstract

yoUR Feedback Hub is a full-stack web application designed to modernize and streamline feedback and grievance management in educational institutions and organizations. Traditional methods—paper forms, unmonitored suggestion boxes, and informal emails—lack traceability, accountability, and structured data, undermining institutional trust and limiting actionable insights. The platform closes this gap by providing a unified digital environment where every submission, from initial registration to final resolution, is logged, tracked, and managed through transparent, role-based workflows. Its dual-interface design includes a submitter portal for students and staff to lodge feedback, file grievances, and monitor progress in real time, alongside an administrator dashboard offering analytics, case assignment, and reporting tools. Built with React, Node.js, and Firebase, the system ensures secure cloud authentication, real-time database synchronization, and a responsive interface that works seamlessly across devices without compromising usability or performance.

2. Keywords

Feedback Management System, Grievance Redressal, React, Node.js, Firebase, Real-Time Tracking, User Authentication, Digital Forms, Web Application, Campus Services, Data Analytics, Automated Status Tracking, Responsive Design, Institutional Governance, Cloud Infrastructure.

3. Synopsis

The yoUR Feedback Hub project was developed to address a common challenge in educational institutions: the lack of a structured, transparent, and digitally integrated channel for students, faculty, and staff to raise concerns, register grievances, and receive timely responses. Traditional informal methods often lead to inconsistent handling, long delays, and no visibility for the complainant, eroding trust. This platform replaces that fragmented system with a secure, scalable, and workflow-driven digital environment that manages submissions end-to-end.

The system supports two main user roles. End users—students and employees—interact with a streamlined, category-based interface to submit feedback or formal grievances. Each submission receives a unique reference ID and enters a monitored workflow. Administrators use a dedicated dashboard to view submissions, access visual analytics, assign cases to appropriate personnel, and deliver formal responses directly to the submitter. Real-time notifications via Firebase Realtime Database ensure instant updates, eliminating manual refreshes and unnecessary follow-ups. The platform is designed to reduce resolution times, prevent submission loss, and foster institutional accountability and responsiveness.

4. Technologies Used

Frontend: React.js — A declarative, component-based JavaScript library used to construct a modular and fully responsive user interface. React's virtual DOM diffing mechanism ensures highly efficient rendering across both desktop and mobile browsers.

Backend: Node.js — A non-blocking, event-driven JavaScript runtime environment that serves as the application's server layer. Its asynchronous I/O model enables concurrent handling of multiple user sessions without performance degradation.

Backend-as-a-Service: Firebase — Google's integrated BaaS platform provides real-time NoSQL database synchronisation, OAuth-based cloud authentication, secure data storage, and serverless function execution, collectively eliminating the need for traditional server management.

Other Tools: Firebase Cloud Messaging (FCM) for push notifications; Firebase Hosting for deployment; CSS3 with responsive grid frameworks for adaptive UI layout; npm for package and dependency management.

5. Procedure

Step 1: User Registration and Authentication

Users register with their institutional email, validated via Firebase Authentication using OAuth 2.0. A profile is created in Firestore, and a session token grants access to the submission portal and personal dashboard.

Step 2: Feedback or Grievance Submission

Authenticated users select a category—general feedback, academic grievance, infrastructure complaint, or service failure—on a structured digital form. Mandatory fields ensure submissions contain sufficient detail for processing without follow-up.

Step 3: Unique Reference Assignment and Workflow Initiation

Each submission receives a unique alphanumeric reference. The backend writes the record to Firebase Realtime Database, triggers the workflow engine, sets the status to “Received,” and notifies the user.

Step 4: Administrator Review and Case Assignment

Submissions appear in real-time on the admin dashboard. Staff review details, assign cases to the appropriate department, and update the status to “Under Review,” with notifications sent to the assigned personnel and the submitter.

Step 5: Resolution, Response Dispatch, and Closure

Administrators log resolution notes, document actions, and send a formal response via the dashboard. The system updates the case to “Resolved,” timestamps closure, delivers the response to the submitter, and archives the record for auditing and trend analysis.

6. Why It Is Best

yoUR The Feedback Hub sets itself apart from traditional feedback systems and generic ticketing tools through technical sophistication, institutional focus, and user-centered design. Unlike paper-based or email-dependent approaches, it offers full end-to-end traceability, with every submission, status change, internal note, and administrator response timestamped and immutably recorded, creating a verifiable audit trail that supports accountability and quality assurance. Real-time synchronization via Firebase ensures both submitters and administrators always access up-to-date case information, eliminating delays from manual updates or batch processing. Its dual-interface architecture separates the end-user experience from the administrator workflow, reducing cognitive overload while giving each user role only the data and functionality they need. Embedded analytics transform raw submissions into actionable institutional intelligence, revealing complaint trends, departmental bottlenecks, seasonal spikes, and resolution benchmarks for evidence-based decision-making. A responsive, device-agnostic design ensures consistent functionality across smartphones, tablets, and desktops, broadening accessibility across diverse institutional populations.

7. Conclusion

yoUR Feedback Hub is a technically robust solution that replaces unstructured, ad hoc communication channels with a unified, transparent, workflow-driven platform, addressing administrative delays and institutional distrust. Built with React, Node.js, and Firebase, it is real-time, scalable, and capable of handling high submission volumes without dedicated server management. Features like structured forms, automated reference tracking, role-based case assignment, real-time notifications, and visual analytics ensure every submission is logged, tracked, and receives a documented response. The platform reduces resolution times, minimizes follow-up inquiries, strengthens data integrity, and improves communication between students, staff, and administrators. Its modular design supports AI-powered categorization, priority scoring, multilingual support, expanded notifications, advanced role management, anonymous submissions, and a mobile companion app, making yoUR Feedback Hub a comprehensive governance platform for institutions of any size.

Virtual Study Group Platform

A Full-Stack, Real-Time, AI-Augmented Collaborative Learning Web Application

ABSTRACT

The The Virtual Study Group Platform is a full-stack web application designed to replicate and enhance collaborative in-person study experiences for remote learners. Built on a three-tier architecture using Node.js, Express.js, MongoDB, and React with TypeScript, it enables students to form study groups, conduct live sessions, and collaborate on a real-time shared whiteboard powered by Socket.IO. An integrated OpenAI API assistant provides contextually aware support through personalised study plans, concept explanations, spaced-repetition quizzes, and post-session summaries. Security is maintained via JWT authentication and role-based access control, while a gamification layer with streak tracking and achievement badges sustains long-term engagement. A personal analytics dashboard converts study activity into measurable performance insights, empowering students to reflect on and improve their learning habits.

KEYWORDS

Collaborative Learning, Full-Stack Web Application, Real-Time Communication, AI Study Assistant, Socket.IO, React, TypeScript, Node.js, Express.js, MongoDB, JWT Authentication, Role-Based Access Control, Gamification, Learning Analytics, OpenAI API, WebSocket, Spaced Repetition, Collaborative Whiteboard.

SYNOPSIS

The Most digital learning tools confine students to isolated, individualised tracks that fail to replicate the collaborative engagement, peer accountability, and social interaction central to effective in-person study. The Virtual Study Group Platform was designed to close this gap by offering a feature-complete environment where students can form study communities, conduct live sessions, and access AI-powered academic support that is contextually aware of their objectives and progress. The Node.js and Express.js backend manages all API logic, authentication, and real-time event orchestration, while MongoDB persists user profiles, group data, session records, and AI interaction histories in a schema-flexible document model. The React with TypeScript frontend delivers complex role-specific views, and Socket.IO establishes persistent WebSocket connections powering instant messaging, live presence indicators, and the collaborative whiteboard. The OpenAI API assistant is structurally embedded into group workflows rather than added as a peripheral feature, surfacing study plans, explanations, quizzes, and post-session summaries at contextually appropriate moments. The gamification subsystem and analytics dashboard complete the platform by transforming study effort into a measurable, rewarding, and data-informed academic practice.

TECHNOLOGIES

- **Node.js & Express.js** — Node.js serves as the backend runtime that handles multiple simultaneous requests without slowing down, while Express.js organises all routes, middleware, and API endpoints in a clean and structured manner.
- **MongoDB** — A flexible NoSQL database used to store all platform data including user profiles, group details, session records, and AI interaction logs in JSON-like documents that easily adapt to changing data requirements.
- **React with** TypeScript — React builds the user interface using reusable components, while TypeScript adds type safety to the codebase, resulting in a reliable and maintainable frontend that handles complex views like dashboards, workspaces, and analytics panels.
- **Socket.IO** — A library that enables real-time, two-way communication between the browser and server, used to power live chat, online presence indicators, and the collaborative whiteboard with very low latency across all browsers.
- **OpenAI API** — Powers the AI study assistant, which uses each group's subject context and the student's progress data to generate personalised study plans, concept explanations, practice quizzes, and post-session summaries on demand.
- **JWT Authentication** — A token-based login system where a signed token is issued at login and verified on every request, removing the need for server-side session storage while ensuring that each user can only access the features and data permitted by their assigned role.

PROCEDURE

1. **User Registration, Authentication, and Profile Initialisation**: New users register through the React frontend, which sends their credentials to the Express.js backend. The backend hashes the password using bcrypt, saves the user record in MongoDB, and returns a signed JWT. On future logins, this token is attached to every API request and validated by Express.js middleware before granting access. Once authenticated, the user's profile, group memberships, streak data, and achievements are loaded onto their dashboard.
2. **Study Group Creation and Member Management:** Users create study groups by providing a subject area, description, and privacy settings, which the backend stores as a MongoDB document with the creator set as group owner. Other users can find and join groups via search or invite links, with the backend verifying role-based permissions before granting membership. Owners and moderators can manage members, assign roles, and update settings, with all changes pushed to connected users in real time through Socket.IO.
3. **Real-Time Session Collaboration and Whiteboard Interaction:** When a session starts, Socket.IO opens a persistent WebSocket connection for all participants, enabling instant group chat and live presence indicators. The collaborative whiteboard streams every drawing and annotation action through Socket.IO rooms, keeping all participants' screens perfectly in sync. All session activity — messages, whiteboard snapshots, and participant events — is continuously saved to MongoDB for later review and AI analysis.
4. **AI Study Assistant Interaction and Contextual Support:** Users query the AI assistant at any time, and the system builds a context-enriched prompt using the group's subject, session agenda, shared resources, and the student's progress before calling the OpenAI API. The response — a concept explanation, quiz set, study plan, or session summary — is displayed in the assistant panel and optionally saved to the group's shared history. Quiz performance is tracked over time and written back to MongoDB to sharpen future AI recommendations.
5. **Progress Tracking, Gamification Rewards, and Analytics Visualisation:** Study time is logged automatically throughout each session, and upon session close the backend updates the user's duration record and checks whether streak or achievement criteria have been met. Earned badges and streak rewards are saved to the user's profile and delivered as frontend notifications. The analytics dashboard then pulls the user's full metrics — study hours, session frequency, topic coverage, quiz trends, and streak history — and renders them as interactive charts to support self-reflection and routine improvement.

WHY IT IS BEST

- **Real-Time Collaboration with Persistent Storage** — Socket.IO delivers instant live communication during sessions, while MongoDB ensures that all messages, whiteboard states, and AI interactions are saved and retrievable at any time. This means students get the energy of a live shared workspace without losing any of their work or history after the session ends.
- **Intelligent, Context-Aware AI Assistant** — Unlike basic chatbots that answer questions in isolation, the OpenAI-powered assistant understands each group's subject, session history, and individual student progress to deliver study plans, explanations, and quizzes that are directly relevant to what the student is currently working on. It also uses spaced repetition and active recall — two scientifically proven memory techniques — built naturally into the study workflow without requiring any extra tools.
- **Gamification for Consistent Study Habits** — Streak tracking, achievement badges, and study milestones apply behavioural reinforcement principles to keep students motivated and accountable over time. This directly addresses one of the biggest challenges in online learning — the drop in motivation without external structure — by turning regular study into a rewarding and progress-driven habit.
- **Scalable, Inclusive, and Accessible Design** — The clean separation between the Node.js backend and React frontend, along with MongoDB's flexible data model, makes the platform easy to scale and extend as usage grows. Its location-agnostic, timezone-flexible architecture ensures that any student, anywhere in the world, can access high-quality collaborative learning regardless of their institution or whether a local peer group is available to them.

CONCLUSION

The Virtual Study Group Platform successfully unifies real-time communication, AI-powered academic assistance, gamified engagement, and personal learning analytics into a single, production-quality web application. Built on a three-tier architecture using Node.js, Express.js, MongoDB, and React with TypeScript, it delivers the scalability and performance required to support concurrent multi-group sessions at institutional scale. Socket.IO enables instant messaging, live presence, and synchronised whiteboard collaboration, while the OpenAI API assistant provides contextually aware study plans, explanations, quizzes, and post-session reports tailored to each student's progress. JWT authentication and role-based access control enforce consistent security across all system layers, and the gamification and analytics subsystems transform study effort into a measurable, rewarding practice. Future enhancements include WebRTC-based video and audio conferencing, adaptive learning pathways driven by accumulated quiz and session data, and a React Native mobile application for iOS and Android. Integration with learning management systems such as Moodle and Canvas will embed the platform within existing academic workflows, while multilingual support and real-time message translation will broaden accessibility for international student cohorts. AI-assisted moderation tools and structured peer review workflows will further support deployment in formal institutional contexts requiring active academic integrity governance. Together, these planned extensions will evolve the platform into a fully integrated, globally accessible, and institutionally deployable collaborative learning ecosystem.

VitalGuard AI — Smart Attendance & Health Monitoring System

An AI-Driven Biometric Attendance and Real-Time Wellness Assessment Platform Using Facial Recognition and Vital Sign Integration

ABSTRACT

VitalGuard AI is an intelligent system for employee attendance and workplace health monitoring. It replaces manual registers and RFID cards with biometric facial recognition via Google Gemini AI, ensuring only the present employee can mark attendance. The platform also monitors health by analysing facial expressions for fatigue, stress, and mood, while collecting real-time sensor data for heart rate, body temperature, and blood oxygen levels. Each measurement is compared against medical thresholds to compute a health risk score. This enables early detection of potential health issues and supports timely intervention. The system provides a unified, browser-accessible interface for seamless attendance and wellness management.

KEYWORDS

Smart Attendance, Health Monitoring, Facial Recognition, Google Gemini AI, Geofencing, Vital Signs, SpO2, Heart Rate, Body Temperature, React, TypeScript, Vite, Anomaly Detection, Wellness Analytics, MediaDevices API, Geolocation API, Biometric Authentication, Health Risk Score.

SYNOPSIS

VitalGuard AI consolidates the full attendance and health monitoring lifecycle into a single browser-accessible platform, serving both employee self-service and administrative oversight workflows through dedicated, role-specific interface components. During each check-in event, the system simultaneously performs facial biometric identification, geofenced location validation, AI-driven wellness inference from facial expression analysis, and real-time acquisition of vital sign sensor readings for heart rate, body temperature, and blood oxygen saturation. These physiological data streams are evaluated against medically established normal ranges to compute a composite health risk score for each employee interaction, with values outside acceptable thresholds triggering automated anomaly alerts to the administrative dashboard. HR managers and safety officers access consolidated attendance records, absenteeism pattern visualisations, aggregated team wellness metrics, and longitudinal health trend charts, empowering data-driven decisions on workload adjustment, medical referral, and occupational safety compliance across the entire organisation.

TECHNOLOGIES

Frontend Framework: React with TypeScript — declarative, component-based single-page application architecture with static type safety, delivering robust and maintainable interfaces for both employee and administrator workflows.

Build Tooling: Vite — high-performance frontend build tool providing near-instantaneous development server startup and optimised, tree-shaken production bundle generation.

AI & Biometrics: Google Gemini AI via the official Generative AI SDK — performs facial recognition for biometric employee identification and simultaneously infers wellness indicators including fatigue, mood, stress, and energy levels from captured images.

Camera Access: MediaDevices API — browser-native permission-gated video capture interface enabling secure, real-time camera stream acquisition for facial recognition and wellness image analysis.

Location Validation: Geolocation API — retrieves real-time GPS coordinates and compares them against administrator-configured authorised workplace boundary polygons to enforce geofenced attendance marking.

Health Sensor Integration: Real-time vital sign acquisition module — ingests readings from connected health sensors measuring heart rate (BPM), body temperature (°C), and blood oxygen saturation (SpO2 %) and evaluates each against medically established normal thresholds.

Analytics Dashboard: React-based interactive visualisation layer — renders attendance records over configurable date ranges, health risk score aggregations, anomaly detection alerts, and longitudinal employee wellness trend charts for administrative review.

PROCEDURE

Step 1: Employee Onboarding and Baseline Setup

New employees are registered with facial biometric images via the MediaDevices API and a baseline health profile including heart rate, body temperature, and SpO_2. These reference points serve as the standard for future check-ins and wellness monitoring.

Step 2: Check-In Data Capture
During check-in, employees grant camera and geolocation access. Live facial images and GPS coordinates are captured simultaneously, providing inputs for identity verification, health assessment, and location validation.

Step 3: AI Biometric and Health Analysis
Facial images are processed by Google Gemini AI to verify identity and assess wellness indicators such as fatigue, stress, and emotional state. Peripheral sensors measure heart rate, temperature, and SpO_2, with any deviations flagged for risk assessment.

Step 4: Risk Scoring and Geofence Validation
The system calculates a composite health risk score by combining AI wellness insights and sensor deviations, classifying it as low, moderate, or elevated. GPS coordinates are checked against authorised geofences to prevent off-site check-ins, and alerts are triggered for any anomalies.

Step 5: Feedback and Administrative Oversight
Employees receive personalised check-in summaries with attendance status, vital signs, risk tier, and AI-generated recommendations. Administrators use a dashboard to monitor attendance, health trends, geofence compliance, and flagged alerts for informed decision-making.

WHY IT IS BEST

Proxy-Resistant Biometric Verification: Facial recognition powered by Google Gemini AI ensures attendance is recorded exclusively by the physically present, biometrically verified individual, eliminating the proxy marking and card sharing vulnerabilities of all legacy mechanisms.

Proactive Health Surveillance: Real-time vital sign monitoring and AI-driven facial wellness inference detect physiological deterioration at the point of check-in, enabling early intervention before health conditions escalate into serious workplace incidents.

Unified Single-Platform Architecture: Combining attendance tracking, geofence validation, biometric identification, and health monitoring within one browser-accessible application eliminates the operational fragmentation of managing separate systems for each function.

Zero Installation Accessibility: Delivered as a browser-based web application, the platform requires no native application installation or device-specific software, ensuring immediate accessibility across any modern desktop or mobile device with a camera.

Data-Driven HR Decision Support: The administrative analytics dashboard aggregates attendance records, health risk scores, anomaly alerts, and longitudinal wellness trends into actionable visualisations that empower evidence-based HR and occupational safety management.

Extensible and Regulation-Ready Architecture: The modular React and TypeScript codebase provides a clean foundation for integrating wearable device data streams, cloud persistence, predictive AI health modelling, and occupational health compliance reporting modules.

CONCLUSION

VitalGuard AI integrates facial recognition, AI-powered wellness analysis, geolocation validation, and real-time health sensor monitoring into a single browser-based platform, replacing vulnerable manual and RFID-based attendance systems with biometrically verified, geofenced check-ins. It combines facial wellness indicators with heart rate, body temperature, and SpO_2 readings to proactively detect at-risk employees and enable early intervention. Built with React, TypeScript, Vite, and Google Gemini AI, the platform is scalable, maintainable, and validated across core attendance and health workflows. Planned enhancements include wearable device integration for continuous monitoring, cloud-hosted multi-session data persistence, predictive AI health risk models, multi-site geofencing, automated HR alert triggers, and compliance-ready reporting, establishing VitalGuard AI as a comprehensive enterprise workforce wellness and attendance solution.

URCET Campus Chatbot

An AI-Powered Conversational Assistant for Usha Rama College of Engineering and Technology

1. Abstract

The URCET Campus Chatbot is an AI-powered web assistant designed for Usha Rama College of Engineering and Technology, providing a single, natural language interface to access campus information. It consolidates fragmented data on academic programs, faculty, admissions, fees, exams, and policies, reducing repetitive queries for staff and streamlining information access for students, parents, and applicants. The hybrid response system intelligently combines a curated knowledge base for routine queries with Google Gemini 2.5 Flash AI for complex or contextual questions. Built with React 19, TypeScript, and Vite, and integrated via the Google Generative AI SDK, the platform ensures a responsive, accessible, and performant experience across desktop and mobile devices without complex backend infrastructure.

2. Keywords

Campus Chatbot, Generative AI, Gemini 2.5 Flash, React 19, TypeScript, Vite, Conversational Interface, URCET, Campus Information System, Hybrid Response Architecture, Google Generative AI SDK, Natural Language Processing, Smart Query Processing, Context-Aware Conversation, Responsive Design.

3. Synopsis

The rapid digitisation of higher education has raised expectations for immediate, accurate, and 24/7 access to institutional information, which traditional channels cannot meet. Campus data scattered across portals, notice boards, and printed documents burdens users and administrative staff with repetitive queries. The URCET Campus Chatbot addresses this with a unified AI-powered interface, trained on a comprehensive knowledge base covering all aspects of campus life at Usha Rama College of Engineering and Technology. Its hybrid response system directs routine, high-frequency queries to a curated knowledge base for guaranteed accuracy, while more complex or contextual queries are escalated to Google Gemini 2.5 Flash via the Generative AI SDK. A fallback mechanism ensures uninterrupted service even if the AI model is unavailable. Quick-action buttons provide fast access to departments, faculty contacts, exams, facilities, and policies. Built with React 19 and TypeScript, the responsive interface delivers a consistent experience across desktop, tablet, and mobile devices, ensuring equitable access for all users.

4. Technologies Used

Frontend - React 19 with TypeScript – Modular, maintainable interface with type safety to prevent runtime errors.

Build Tool: Vite – Fast development server and optimized production bundles for quick loading.

AI Model: Google Gemini 2.5 Flash – Generates real-time, contextual responses to complex queries.

Hybrid Response Engine: Curated knowledge base + AI fallback – Answers common questions instantly and escalates unusual queries to AI.

Query Processing: Smart module – Understands different phrasings and correctly routes user questions.

Interface Design: Mobile-first responsive layout with quick-access buttons for frequently requested info.

5. Procedure

Step 1: Access and Setup:

Users open the chatbot on any device with no login needed. The interface loads, quick-access buttons appear, and the system connects to the AI backend.

Step 2: Submit Query:

Users type questions naturally. The system interprets intent and standardizes phrasing so different ways of asking the same thing are understood correctly.

Step 3: Answer Retrieval:

The system first checks the curated knowledge base for a matching answer. If none is found, it sends the query to Google Gemini AI for a real-time response. A fallback ensures continuity if the AI is unavailable.

Step 4: Context-Aware Conversation:

The chatbot remembers previous messages, allowing follow-up questions to be understood correctly in context. Multi-turn interactions stay coherent.

Step 5: Display Response:

Answers appear in the chat interface with timestamps and formatting. The conversation thread preserves history, keeping the session continuous and easy to review.

6. Why It Is Best

Hybrid Knowledge Architecture: The URCET Campus Chatbot integrates a curated knowledge base for high-frequency, high-stakes queries with the Gemini 2.5 Flash AI model for complex, context-dependent questions, ensuring both factual accuracy and generative flexibility.

Reliable and Continuous Service: The fallback mechanism guarantees uninterrupted access even if the AI service is temporarily unavailable, while the knowledge base provides instant, latency-free responses to predictable queries, maintaining trustworthiness for critical campus information like fees and deadlines.

Intuitive, Accessible Interface: Built with React 19 and TypeScript, the frontend delivers a responsive, mobile-friendly experience with quick-action buttons and smart query processing, allowing users to interact naturally without needing precise phrasing or prior knowledge of the system.

Efficient, Secure, Low-Overhead Deployment: By avoiding complex backend infrastructure, the chatbot reduces operational cost, security exposure, and maintenance overhead, while still enabling rapid updates to institutional content and ensuring a scalable, maintainable solution for campus-wide information delivery.

.

7. Conclusion

The URCET Campus Chatbot demonstrates that institution-specific AI assistants can be efficiently developed and deployed without complex backend infrastructure, providing a reliable, production-quality conversational interface for students, faculty, parents, and prospective applicants. By combining a curated knowledge base with Gemini 2.5 Flash AI and a robust fallback mechanism, the system balances factual accuracy with generative flexibility. Smart query processing, context-aware multi-turn conversation management, quick-action navigation, and a responsive device-agnostic interface deliver a superior user experience while reducing administrative burden and improving information accessibility. The current implementation establishes a foundation for next-generation enhancements, including real-time database integration, regional language support (e.g., Telugu), voice interaction, and an administrative analytics dashboard. Multi-campus deployment capabilities and live-synchronised knowledge updates will expand scalability, accessibility, and operational efficiency. Collectively, these improvements position the platform as a scalable, data-driven, linguistically inclusive institutional intelligence solution suitable for modern higher education administration.

Talent Sphere

A Virtual Career Fair Networking Portal with AI-Assisted Recruitment and Real-Time Queue-Based Video Interaction

ABSTRACT

The traditional in-person career fair connects students with employers but is limited by geography, time, cost, and low interaction efficiency. Existing online platforms only replicate static job portals, offering little real-time engagement or process visibility. **Talent Sphere** is a virtual career fair platform that recreates and enhances this experience online, connecting candidates and recruiters through interactive, queue-based video interviews instead of static submissions. The system supports three roles—candidates, recruiters, and administrators—each with dedicated dashboards and tailored workflows. A client-side mock backend simulates database persistence, real-time queue updates, notifications, and network latency, enabling a fully self-contained, deployable demonstration of complex recruitment workflows.

KEYWORDS

Virtual Career Fair, Recruitment Portal, React, TypeScript, Vite, Tailwind CSS, Google Gemini AI, PDF.js, Role-Based Access Control, Event-Driven Architecture, Pub-Sub Pattern, WebRTC, Queue-Based Video Interview, Single Page Application, AI Resume Analysis, Mock Backend.

SYNOPSIS

Talent Sphere centres its engagement model on direct, synchronous human interaction rather than asynchronous form filling, delivering a fundamentally more dynamic and equitable recruitment experience across three role-specific interfaces. Candidates upload PDF resumes that are parsed client-side using PDF.js and enriched by Google Gemini AI for automated skill identification and profile structuring, before exploring virtual company booths, joining interview queues, and receiving real-time push notifications as their queue position advances. Recruiters access a comprehensive dashboard to monitor queue status, admit candidates into mock video interview sessions, and record structured post-interaction evaluations for post-fair review. Administrators govern platform integrity by validating recruiter registrations through a secret key verification mechanism, moderating user-generated conduct reports, and monitoring overall activity to maintain a safe and professional virtual fair environment. The event-driven, publish-subscribe architecture implemented entirely on the client side enables real-time state propagation across all role interfaces without requiring any deployed server infrastructure.

TECHNOLOGIES

Frontend Framework: React with TypeScript — declarative, component-based single-page application architecture with static type safety for maintainable, large-scale UI development.

Build Tooling: Vite — next-generation frontend build tool delivering near-instantaneous development server cold starts and optimised, tree-shaken production bundling.

Styling: Tailwind CSS — utility-first CSS framework enabling rapid, consistent, and fully responsive interface styling across all three role-specific dashboard layouts.

AI Integration: Google Gemini AI — processes extracted resume text to perform automated skill identification, candidate profile enrichment, and structured capability mapping without manual data entry.

PDF Processing: PDF.js — client-side PDF parsing library that extracts raw text content from candidate-uploaded curriculum vitae files entirely within the browser environment.

Mock Backend: Client-side event-driven pub-sub system — simulates database persistence, queue state transitions, notification dispatch, and artificial network latency without any server deployment.

Architecture Pattern: Role-Based Access Control (RBAC) with modular component separation — enforces distinct capability boundaries across candidate, recruiter, and administrator tiers throughout all platform layers.

PROCEDURE

Step 1: User Registration and Authentication

New users select their role—candidate, recruiter, or administrator—and complete the role-specific registration form. Recruiter accounts remain pending until an administrator verifies legitimacy via a secret key, ensuring only authorised organisations access recruiter tools. Upon login, users are directed to their role-specific dashboards, providing tailored tools and information.

Step 2: Candidate Profile Creation and AI Analysis
Candidates upload their CVs in PDF format, which are parsed on the client side using PDF.js. The extracted text is analysed by Google Gemini AI to automatically identify skills, competencies, and qualifications, enriching the candidate profile without manual data entry.

Step 3: Virtual Career Fair and Queue Management
Candidates explore interactive company booths displaying organisational profiles, roles, and recruiter details. They join interview queues with a single click, while the platform updates queue positions and recruiter dashboards in real time using a client-side event system.

Step 4: Interview Sessions and Feedback Collection
Recruiters admit candidates from the queue to mock video interview sessions simulating real-time one-on-one interactions. After interviews, recruiters complete evaluation forms, and candidates submit feedback, creating a continuous quality improvement loop for post-fair assessment.

Step 5: Administrative Oversight and Governance
Administrators manage pending recruiter registrations, approve or reject accounts based on verification, and moderate conduct reports to ensure professional integrity. They maintain platform safety and fairness across all active virtual career fair sessions.

WHY IT IS BEST

Live Interaction Over Static Submissions: Queue-based video interview sessions replace opaque application pipelines with real-time, human-centred conversations, replicating the immediacy and engagement of physical career fair booth interactions in a fully online environment.

AI-Powered Resume Intelligence: Google Gemini AI automatically extracts and structures candidate skills from uploaded PDF resumes, eliminating manual profile entry and delivering a richer, more accurate representation of candidate capabilities to recruiters.

Zero Server Infrastructure Required: The client-side pub-sub mock backend enables complete demonstration of real-time queue management, role-based access control, and notification delivery without requiring any deployed server, database, or cloud infrastructure.

Three-Role Purpose-Built Experience: Dedicated dashboards and workflows for candidates, recruiters, and administrators ensure every user type operates within an interface precisely tailored to their responsibilities, eliminating role confusion and interface bloat.

Geographically Unrestricted Access: The fully online platform removes the geographical, logistical, and financial barriers of physical career fairs, enabling candidates and recruiters from any location to participate in a structured, high-quality recruitment event.

Production-Ready Extensible Architecture: The modular, component-based design and event-driven architecture provide a clean and well-structured foundation for integrating WebRTC video, cloud databases, and secure authentication in a future production deployment.

CONCLUSION

Talent Sphere demonstrates that the interactive, human-centred engagement of a physical career fair can be authentically replicated and enhanced online, overcoming the limitations of traditional events and static recruitment portals. By focusing on synchronous, queue-based video interactions instead of asynchronous form submissions, the platform delivers a more transparent and equitable experience for candidates, recruiters, and administrators. Its client-side architecture incorporates a pub-sub mock backend, Google Gemini AI resume analysis, PDF.js processing, role-based access control, and real-time notifications, forming a cohesive, self-contained prototype. The development roadmap includes WebRTC-powered live video sessions, migration to cloud-hosted databases for true persistence and concurrency, secure OAuth 2.0 or JWT authentication, and advanced recruitment analytics. Additional enhancements like calendar-based interview scheduling, automated reminders, and customizable virtual booths will further improve user experience. Together, these upgrades position Talent Sphere as a scalable, production-ready, next-generation virtual recruitment platform.

React Single Page Application Using Create React App

Front-End Development, Toolchain Configuration, Testing, and Production Deployment

ABSTRACT

Modern web application development demands user interfaces that are responsive, maintainable, and capable of delivering a seamless experience across devices and varying network conditions. As the complexity of front-end systems has grown, so too has the sophistication of the tooling required to build them effectively. Configuring a React project from scratch—integrating Babel for transpilation, Webpack for module bundling, ESLint for code quality enforcement, Jest for testing, and optimised build pipelines for production deployment—is a non-trivial undertaking that introduces significant friction, particularly for beginners and small development teams working under time constraints. To address these challenges, Create React App (CRA) was introduced as a command-line interface tool that abstracts away manual project configuration by providing a fully pre-configured, production-ready development environment with a single command. This project investigates the design, development, testing, and deployment of a React Single Page Application bootstrapped using CRA, treating the application not merely as a functional deliverable but as a structured evaluation of the CRA toolchain as a foundation for professional front-end development, examining its capabilities, trade-offs, workflow conventions, and positioning relative to modern alternatives such as Vite and Next.js.

KEYWORDS

React, Create React App, Single Page Application, Webpack, Babel, npm Scripts, Front-End Tooling, JavaScript, TypeScript, Jest, React Testing Library, Build Optimisation, Code Splitting, Lazy Loading, Vite, Next.js, CSS Modules, React Router.

SYNOPSIS

The application demonstrates the full breadth of modern front-end development practice within the CRA environment, covering component-based UI design using functional React components and hooks, client-side routing via React Router, local and global state management strategies, and production deployment to static hosting platforms. Performance optimisation through code splitting and lazy loading is explored alongside CRA's four primary npm workflow scripts: npm start for live-reloading development, npm test for Jest-powered watch-mode testing, npm run build for minified production bundle generation, and npm run eject for exposing the underlying Webpack and Babel configuration for advanced customisation. The project also situates CRA within the broader front-end tooling landscape by comparing it against traditional manual Webpack setups and modern alternatives including Vite and Next.js, evaluating each across build speed, configuration flexibility, ecosystem support, and suitability for projects of varying scale and complexity.

TECHNOLOGIES

Core Framework: React — component-based JavaScript library for building declarative, virtual DOM-powered user interfaces with functional components and hooks.

Project Scaffolding: Create React App (CRA) — zero-configuration CLI tool providing a pre-configured development environment, build pipeline, and test runner out of the box.

Transpilation & Bundling: Babel and Webpack — Babel transpiles modern JavaScript and JSX to browser-compatible ES5; Webpack bundles, tree-shakes, and optimises all module dependencies.

Routing: React Router — declarative client-side routing library enabling multi-page navigation within the Single Page Application without full page reloads.

Testing: Jest and React Testing Library — Jest provides the test runner and assertion framework; React Testing Library enforces behaviour-driven component testing aligned with user interactions.

Build Analysis: Lighthouse and Source Map Explorer — Lighthouse profiles runtime performance and accessibility; Source Map Explorer analyses production bundle composition for size optimisation.

Styling: CSS Modules — scoped, locally namespaced stylesheets preventing class name collisions and enforcing style encapsulation at the component level.

PROCEDURE

Step 1: Project Setup and Component Design

The project is initialized using Create React App (CRA), which sets up a ready-to-run React environment with

Webpack, Babel, ESLint, Jest, and a development server. The component hierarchy is then planned, mapping pages and UI regions into reusable functional components with clear separation of concerns for presentational, container, and utility responsibilities.

Step 2: Component Implementation and Styling

Functional components are built using React hooks—useState for local state and useEffect for lifecycle effects—each co-located with its CSS Module and Jest test file. This ensures modular, self-contained development and avoids global style conflicts or cross-component dependencies.

Step 3: Routing and Code Splitting

React Router is configured with a centralized route map linking URLs to page-level components. Route-based code splitting using React.lazy and Suspense ensures that page bundles are loaded on demand, reducing initial load times and improving performance for users on slower connections.

Step 4: API Integration and Testing

External API interactions are abstracted into custom hooks managing data fetching, loading states, and error handling. Components consume data only via these hooks, maintaining separation of UI and business logic. Parallel Jest and React Testing Library suites validate user-facing behavior, including rendering, interactions, and conditional displays, rather than internal implementation details.

Step 5: Performance Optimization and Deployment

Production builds are created with npm run build, generating minified, tree-shaken, hashed assets. Performance profiling using Google Lighthouse and Source Map Explorer identifies optimization opportunities. The build is then deployed to a static hosting platform, completing the full development lifecycle from scaffolding to live production with zero manual server configuration.

WHY IT IS BEST

Zero Configuration Onboarding: CRA eliminates the need to manually configure Babel, Webpack, and ESLint, enabling developers to begin writing production-quality React code immediately without any toolchain setup overhead.

Standardised Workflow: The four npm scripts — start, test, build, and eject — provide a universally consistent development workflow that is immediately familiar to any developer with prior React experience.

Built-In Testing Infrastructure: Jest and React Testing Library are pre-integrated and pre-configured, lowering the barrier to writing and running tests and enforcing a culture of quality from the very first line of code.

Production-Ready Build Pipeline: The npm run build command generates a fully optimised, minified, and cache-busted production bundle without any manual Webpack tuning, producing deployment-ready artefacts with minimal developer effort.

Advanced Customisation Path: The eject option provides a clear and explicit migration path to full manual toolchain control for teams whose requirements outgrow the CRA defaults, without requiring a project rewrite.

Broad Ecosystem Compatibility: CRA's widespread adoption ensures extensive community support, abundant third-party library compatibility, and comprehensive documentation, making it a low-risk and well-supported foundation for new projects.

CONCLUSION

This project demonstrates that Create React App (CRA) provides a practical, well-structured, and productive foundation for developing modern React single-page applications, balancing beginner accessibility with professional-grade development practices. By encapsulating Babel transpilation, Webpack bundling, ESLint enforcement, and Jest testing, CRA allows developers to focus on architecture, component design, and user experience rather than toolchain maintenance. The project confirms that CRA’s defaults support industry best practices, including component encapsulation, behaviour-driven testing, code splitting, and separation of concerns, producing a production-ready application with strong performance, bundle size, and test coverage metrics. Future enhancements could include migrating to Vite for faster build times and hot module replacement, adopting Next.js for server-side rendering and SEO improvements, establishing a CI/CD pipeline via GitHub Actions, expanding testing with Cypress or Playwright for end-to-end coverage, and integrating advanced performance monitoring like Web Vitals and real-user metrics to enable ongoing data-driven optimisation throughout the application’s lifecycle.

ScholarChain

A Blockchain Academic Records with AI Credential Verification

1. Abstract

ScholarChain is a blockchain-based academic record management system that addresses the weaknesses of conventional credential systems, where paper certificates can be forged, centralized databases are vulnerable, and manual verification causes delays. It encodes degrees, transcripts, and course records as digital assets within cryptographically linked blockchain blocks, using SHA-256 hashing to make any modification instantly detectable. Role-based access control ensures Registrars, Instructors, and Students have strictly separated permissions, preventing unauthorized changes. An integrated Google Gemini AI module analyses transcript data to generate performance summaries, skill assessments, and career recommendations. Together, these features make ScholarChain a secure, verifiable, and actionable academic advisory platform.

2. Keywords

Blockchain, Academic Records, Digital Credentials, Proof of Work, SHA-256, React, TypeScript, Vite, Web Crypto API, Excel Import, Google Gemini AI, Role-Based Access Control, Transcript Management, Credential Verification, Immutable Ledger, Cryptographic Hashing.

3. Synopsis

The ScholarChain project fills a critical gap in academic credential verification, allowing employers, graduate schools, and licensing bodies to validate records without contacting institutions. Paper certificates are easily forged, and centralized databases are vulnerable to failure and manipulation, especially in global labour markets. ScholarChain uses blockchain principles—cryptographic record linkage, Proof-of-Work, and hash-based tamper detection—to create a secure, self-contained credential ledger. Built with React and TypeScript, it supports registrars issuing credentials, instructors submitting course data, and students accessing verified records and generating shareable reports. The system leverages the Web Crypto API for SHA-256 hashing, bulk record import via the xlsx library, and Google Gemini AI for transcript analysis and career guidance. Together, these features form a robust, extensible proof-of-concept for secure, verifiable, and scalable academic record management.

4. Technologies Used

1. Frontend: React with TypeScript

- Uses React's declarative component model with TypeScript's type system.
- Builds a maintainable, role-sensitive interface with fewer runtime errors.

2. Build Tool: Vite

- Provides near-instant development server startup and fast hot module replacement.
- Generates highly optimised production bundles for better performance.

3. Cryptographic Engine: Web Crypto API

- Performs SHA-256 hashing for secure block linking.
- Browser-native API avoids third-party dependencies and ensures consistent hashes.

4. AI Integration: Google Gemini AI

- Analyses student transcripts to generate performance summaries.
- Provides skill assessments and personalised career pathway recommendations.

5. Data Import: xlsx Library

- Supports bulk import of Excel files containing student records.
- Allows registrars to process entire cohorts at once.

6. Consensus Mechanism: Proof-of-Work

- Requires solving a computational challenge before adding a new block.
- Ensures that tampering with the blockchain is extremely difficult.

5. Procedure

Step 1: User Authentication and Role Assignment
Users log in to ScholarChain using institutional credentials. The system enforces role-based access control, assigning users to one of three tiers: Registrar, Instructor, or Student. Role profiles determine the interface and restrict data visibility and actions, ensuring students cannot access others' records, instructors see only their courses, and only registrars can issue credentials or interact with the blockchain ledger.

Step 2: Academic Record Submission
Instructors submit student performance data via structured digital forms with mandatory fields and type validation. Registrars can upload bulk Excel files parsed by the xlsx library for high-volume submissions. All records undergo schema validation to confirm required fields—student ID, course code, grade, and academic period—are correctly formatted before blockchain inclusion.

Step 3: Proof-of-Work Block Mining
Once a record is ready, the system initiates the Proof-of-Work process. Using the Web Crypto API, it iteratively computes SHA-256 hashes with candidate nonce values until a valid hash meeting the difficulty criteria is found. The new block, containing record data, timestamp, nonce, and computed hash, is then appended immutably to the chain.

Step 4: Chain Integrity Verification and Tamper Detection
Authorized users or the system can verify chain integrity by recomputing hashes from the genesis block to the current tip, ensuring each block's stored hash matches the computed hash and its predecessor link is valid. Any discrepancy flags tampering, pinpointing the exact block where integrity is compromised.

Step 5: AI-Powered Transcript Analysis and Credential Sharing
Students and registrars can submit transcripts to the integrated Google Gemini AI, which generates insights on academic progress, skills, and career paths. Students can also create shareable credential verification reports, enabling third parties to confirm authenticity via blockchain hash validation.

6. Why It Is Best

ScholarChain stands out from paper-based and generic digital credential systems through cryptographic immutability, decentralised verification, role-based governance, and AI-powered academic insights. Its SHA-256 hash-chained ledger ensures any record tampering is instantly detectable, while Proof-of-Work makes retroactive chain modification computationally infeasible. Role-based access mirrors institutional hierarchies, enforcing strict data partitioning to protect sensitive academic records. Google Gemini AI transforms the system into an active advisory tool, providing personalised performance insights and career guidance. Bulk Excel import enables high-volume record onboarding, making ScholarChain a secure, practical, and institutionally relevant solution for modern credential management.

7. Conclusion

ScholarChain leverages blockchain to address academic credential challenges, featuring a SHA-256 hash-chained immutable ledger, Proof-of-Work consensus, role-based access control, structured and bulk record submission, real-time chain verification, and AI-powered transcript analysis via Google Gemini. Testing confirms tamper detection, strict permission enforcement, and accurate AI-driven insights. The prototype provides a foundation for a production-grade system with planned enhancements like public or permissioned blockchain migration, QR-code credential verification, integration with student information systems, digital signatures, and a mobile credential-sharing app. These improvements eliminate single points of failure, automate data flow, support privacy-preserving credential sharing, and position ScholarChain as a fully interoperable, production-ready platform for institutions of any size.

AR Campus Navigation and Information Portal

A Web-Based Augmented Reality Application for Intelligent Campus Navigation and Information Access

ABSTRACT

Large university and college campuses present a persistent navigation challenge for new students, visiting scholars, prospective applicants, and general visitors. The physical complexity of modern campuses—comprising academic blocks, administrative offices, laboratories, libraries, sports facilities, and event venues spread across expansive grounds—makes spatial orientation a nontrivial task for the uninitiated. Conventional navigation aids such as static printed maps, fixed signboards, and institution website directories address this challenge only partially: they are frequently outdated, incapable of providing real-time directional guidance, and entirely unable to adapt to a user's current position and orientation. The AR Campus Navigation and Information Portal is a web-based Augmented Reality application that transforms an ordinary smartphone into an intelligent, real-time campus guide, overlaying contextual virtual labels, directional cues, and informational annotations directly onto the live camera feed, anchored to real-world building positions through GPS geolocation, device orientation sensors, and browser-native APIs. A conversational AI assistant supplements the AR view for open-ended campus queries.

KEYWORDS

Augmented Reality, Campus Navigation, Progressive Web App (PWA), Geolocation, Google Gemini, AR Overlay, React 19, TypeScript, Tailwind CSS, DeviceOrientation API, Campus Information System, Indoor Navigation.

SYNOPSIS

The platform enables any smartphone user to point their device camera at campus buildings and instantly receive overlaid information: building name, estimated distance, cardinal direction, AI-generated descriptions, and navigation cues—all rendered as augmented reality layers atop the physical environment. An interactive campus map provides a conventional 2D fallback view for users who prefer it, while the indoor navigation module delivers step-by-step guidance for complex multi-floor buildings where GPS signals are unreliable. An administrator dashboard allows authorised staff to manage Points of Interest without developer intervention, keeping the knowledge base current in real time. The system follows a strict privacy-first architecture: no user footage, location history, or personal data is ever stored or transmitted.

TECHNOLOGIES

Frontend Framework: React 19 with TypeScript for a strongly typed, component-driven architecture.

Styling: Tailwind CSS — utility-first responsive design system ensuring consistent cross-device appearance.

Application Architecture: Progressive Web App (PWA) — instant browser access with no app store installation required.

Browser APIs: Geolocation API (GPS positioning), MediaDevices API (camera stream), DeviceOrientation API (compass & tilt).

Generative AI: Google Gemini — powers AI-generated building descriptions and the conversational campus query assistant.

AR Rendering: Browser-native overlay engine anchoring virtual labels to real-world GPS coordinates in real time.

PROCEDURE

1. The user opens the Progressive Web App directly in any modern mobile browser such as Chrome or Safari. No application download, app store installation, or account registration is required at any stage, making the platform immediately accessible to first-time visitors and casual users alike.
2. Upon launch, the application requests permission to access the device camera, GPS location, and orientation sensors through the browser's native permission dialogs. These permissions are essential for the core AR functionality and are handled entirely within the browser environment without any external dependencies.
3. Once permissions are granted, the Geolocation API begins capturing real-time GPS coordinates while the DeviceOrientation API continuously reads the device's compass heading and tilt angle. This sensor data is processed locally on the device to determine the user's precise position and the direction they are facing.

4. The system cross-references the user's live GPS coordinates and viewing angle against the stored campus database of buildings and Points of Interest. Structures that fall within the current field of view are identified and selected for AR overlay rendering in the camera feed.

5. Augmented Reality labels are rendered as floating overlays on the live camera feed, displaying each building's name, estimated distance in metres, and cardinal direction. These overlays dynamically reposition and update in real time as the user moves, rotates, or tilts their device, maintaining accurate spatial anchoring at all times.

6. The user may tap on any AR label to expand a detailed information panel powered by Google Gemini, which provides AI-generated contextual descriptions of the selected building including its departments, facilities, and operating hours. The conversational chat assistant is also available for open-ended natural language queries that extend beyond what is visible on screen.

7. For buildings with complex internal layouts or multiple floors, the indoor navigation module activates automatically. It delivers clear, text-based step-by-step directional instructions guiding the user from the entrance to their destination within the building, compensating for GPS signal degradation in enclosed spaces.

8. Authorised administrators access a dedicated management dashboard to add, edit, or remove campus Points of Interest, building descriptions, and scheduled event information in real time. All content updates are immediately reflected in the user-facing AR experience without any redevelopment or redeployment of the application.

WHY IT IS BEST

Zero Installation Friction: Delivered as a PWA, the application is accessible instantly via any mobile browser, removing the app store barrier entirely for casual visitors and prospective students interacting with the platform for the first time.

Privacy-First by Design: No video footage, photographic data, or location history is ever stored, transmitted, or logged at any point during a session. No user account or login is required, ensuring full functionality for every visitor immediately.

Real-Time AR Guidance: Unlike static maps or fixed signboards, AR overlays update live as users move and reorient, providing genuinely adaptive and position-aware spatial guidance tailored to each user's exact location.

AI-Powered Intelligence: Google Gemini delivers contextually rich building descriptions and handles diverse natural language queries without any custom model training, specialised AI infrastructure, or ongoing model maintenance costs.

Universal Accessibility: Runs on any modern smartphone browser with no dependency on proprietary AR platforms, native SDKs, or complex server-side infrastructure, making it inclusive across a wide range of devices.

Low Institutional Barrier: Built entirely on open web standards, the system significantly lowers deployment and maintenance costs, making it a practical and scalable solution for educational institutions of any size.

CONCLUSION

The AR Campus Navigation and Information Portal represents a major advancement in addressing spatial navigation challenges within large, complex educational campuses. It demonstrates that privacy-conscious augmented reality applications can be built and deployed using standard web technologies alone, without native apps, proprietary AR platforms, or complex server infrastructure, establishing a strong case for browser-based AR as a practical solution for real-world spatial applications. The platform integrates AR overlays, real-time GPS geolocation, device orientation sensing, generative AI, and indoor navigation within a single Progressive Web App, delivering a navigation experience that surpasses traditional static tools while remaining accessible to any smartphone user. Its privacy-first architecture and zero-installation delivery model make it ideal for prospective students, external visitors, and first-time users. The administrator dashboard allows updates without ongoing developer involvement, ensuring sustainable long-term deployment. Future enhancements include live event integration, computer vision-based building recognition, multilingual support, and turn-by-turn pedestrian routing with dynamic path recalculation. Bluetooth beacon-based indoor localisation will further improve navigation accuracy within multi-building campuses. Together, these features create a comprehensive, intelligent, and highly accessible campus navigation ecosystem. By combining accessibility, privacy, and advanced AR functionality, the platform sets a new standard for educational navigation tools, demonstrating that sophisticated, real-world AR applications can be delivered effectively entirely through the web. The system provides a seamless experience for both frequent users and one-time visitors, removing traditional barriers to access and establishing browser-based AR as a practical, scalable, and sustainable solution for modern campus navigation.

College Notice Board

A MERN Stack Web Application for Secure, Centralised, and Targeted Institutional Notice Management

ABSTRACT

Institutional communication within colleges and universities has historically depended upon physical notice boards as the primary medium for disseminating announcements, examination schedules, event notifications, policy updates, and departmental circulars. This traditional model carries substantial limitations that compound as institutions grow in size and complexity: physical notices are spatially restricted to specific campus locations, rendering them inaccessible to students and staff who are off-site or studying remotely. Printed materials offer no mechanism for verifying whether intended audiences have received critical information, and the manual effort required to print, distribute, and replace notices imposes a recurring administrative burden. The College Notice Board is a comprehensive digital notice management system developed using the MERN stack—MongoDB, Express.js, React, and Node.js—that replaces the fragmented, location-dependent physical model with a secure, scalable, and user-friendly web application. The system digitises the entire notice lifecycle from creation and targeted distribution through to automatic expiry and archival, consolidating all institutional communications into a single accessible online platform that significantly improves the speed, reach, and reliability of information delivery across the entire campus community.

KEYWORDS

MERN Stack, College Notice Board, Notice Management System, Web Application, Node.js, Express.js, React, TypeScript, MongoDB, JWT Authentication, Role-Based Access Control, Admin Dashboard, Responsive Design, Digital Institutional Communication.

SYNOPSIS

The platform serves two primary audiences through dedicated interfaces: an administrative panel for authorised staff responsible for managing the full notice lifecycle, and a public-facing interface through which students and stakeholders can view relevant announcements without requiring account registration or login. Administrators can create notices with rich text and image attachments, configure targeted delivery to specific departments or broadcast institution-wide, and set automatic expiry dates to ensure outdated content is removed from the public feed without manual intervention. A role-based dashboard tailors the interface to each user type, while a basic analytics panel provides administrators with visibility into publication history, notice engagement, and system usage patterns. Search and filtering functionalities allow students to locate specific notices rapidly by keyword, department, date range, or category, delivering a significantly superior retrieval experience compared to scanning a cluttered physical notice board.

TECHNOLOGIES

Frontend: React 18 with TypeScript — declarative component-based UI architecture with static typing to minimise runtime errors and support scalable, maintainable development.

Styling: Responsive CSS design system supporting both desktop and mobile viewports, with a switchable dark and light theme for enhanced visual comfort.

Backend: Node.js and Express.js — lightweight RESTful API layer handling all business logic, authentication, and data operations with non-blocking, concurrent efficiency.

Database: MongoDB — flexible document-oriented NoSQL data store accommodating varied data structures for notices, user profiles, departments, and system analytics.

Authentication: JSON Web Token (JWT) — stateless, cryptographically signed session management providing secure, scalable admin authentication without server-side session storage overhead.

Access Control: Role-Based Access Control (RBAC) enforced across both the API layer and frontend interface, restricting all notice management operations exclusively to verified administrators.

PROCEDURE

Step 1: Administrator Authentication

The administrator logs in via a secure portal; credentials are validated against bcrypt-hashed records in MongoDB. A signed JSON Web Token (JWT) is issued for stateless, secure session management, attached to all subsequent API requests as a Bearer token.

Step 2: Dashboard Access and Data Display
Once authenticated, the administrator accesses a role-based dashboard showing active notices, pending drafts, expired entries, and system analytics. The dashboard fetches current data dynamically from the Express.js REST API to ensure accuracy.

Step 3: Notice Creation and Publication
Administrators complete a structured form with the notice title, content, target audience, optional image, and expiry date. Client-side validation ensures correctness before the backend verifies the JWT, processes the notice (including image uploads), stores it in MongoDB, and publishes it to the public student feed.

Step 4: Public Notice Access and Expiry Management
Students and visitors access active notices without logging in. The system filters expired notices automatically on each API call, keeping the feed current while retaining historical records for auditing. Users can search and filter by keywords, department, date, or category.

Step 5: Notice Editing, Deletion, and Analytics
Administrators can edit or delete notices via the dashboard, with edits pre-populating forms and deletions requiring confirmation. An analytics panel tracks publication metrics, departmental distribution, expiry compliance, and peak access, enabling data-driven communication management decisions.

WHY IT IS BEST

Centralised Digital Platform: Consolidates all institutional notices into a single accessible web application, eliminating the spatial restrictions of physical boards and ensuring every announcement reaches students regardless of their physical location.

Zero Friction for Students: The public notice feed requires no account creation, login, or application installation, guaranteeing universal and immediate access for every student, staff member, and campus visitor.

Automated Notice Lifecycle: Configurable expiry dates automate the removal of outdated notices from public view, eliminating manual curation overhead and ensuring the feed always reflects current and relevant institutional information.

Targeted Communication: Department-specific notice routing reduces information overload by ensuring students receive only announcements directly relevant to their academic context rather than undifferentiated institution-wide broadcasts.

Enterprise-Grade Security: JWT authentication with bcrypt password hashing and RBAC middleware ensures all administrative operations are cryptographically protected and role-controlled at every layer of the application stack.

Scalable MERN Architecture: The unified JavaScript MERN stack delivers a cohesive, high-performance full-stack architecture that scales efficiently with growing institutional demand and accelerates ongoing feature development.

CONCLUSION

The College Notice Board demonstrates how modern full-stack web technologies can systematically address longstanding institutional communication challenges, improving operational efficiency, information accessibility, and transparency across campus. By replacing fragmented physical notice boards with a centralized, role-based digital platform, it reduces administrative burden while extending the reach and reliability of every announcement. Built on the MERN stack, React and TypeScript provide a robust, type-safe frontend, Node.js and Express.js deliver a secure API layer, and MongoDB ensures flexible, scalable data persistence to meet evolving institutional needs. JWT-based authentication, bcrypt credential storage, and role-based access control protect administrative functions, producing a polished, production-ready system tailored to both administrators and students. Future enhancements—including email, browser push, and SMS notifications—will proactively alert students to critical notices, while a dedicated mobile app will optimize engagement for on-the-go access on Android and iOS. Advanced analytics will track read rates, departmental trends, and peak access windows, enabling data-driven communication strategies. Multi-institution deployment and integration with Student Information Systems will allow precise audience targeting across campuses, programmes, and academic standings. Collectively, these features establish the College Notice Board as a comprehensive, intelligent, and enduring communication infrastructure capable of supporting the evolving needs of modern educational environments.