iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.80561

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

MediConnect: A Full-Stack Real-Time Hospital Appointment Management System

Prof. Sujata Tirpude[1], Darshan Chudasama[2], Meenakshi Kharat[3]
[1]*Assistant Professor,* [2,3]*Student, Department of Information Technology Engineering, Bharat College of Engineering, Mumbai, India*

Abstract: ***This paper presents the design and implementation of MediConnect, a full-stack, real-time Hospital Appointment Management System constructed as a single-page web application (SPA) following the JAMstack architectural paradigm. The system addresses persistent inefficiencies in manual hospital appointment workflows by delivering a serverless, cloud-native platform. The architecture integrates Supabase for backend-as-a-service (BaaS) — encompassing a PostgreSQL relational database, secure email/password authentication, Row Level Security (RLS), and a WebSocket-driven Realtime engine — with EmailJS for automated browser-triggered notifications and Netlify for static hosting and global content delivery. MediConnect provides two distinct portals: a patient-facing interface enabling self-service registration, specialist browsing, appointment booking, and medical history access; and an administrator dashboard for centralised management of patients, physicians, departments, and operational records. The system achieves sub-second data propagation via WebSocket-based real-time subscriptions, ensuring consistent application state across all concurrent sessions. Experimental results confirm functional completeness, low operational overhead, and a professional-grade user experience, demonstrating that feature-rich healthcare management software can be delivered without conventional server infrastructure.***

Keywords: Hospital Appointment Management, JAMstack, Supabase, Real-Time Web Applications, Single-Page Application, Serverless Architecture, Row Level Security, WebSockets, Healthcare Digitisation

I. INTRODUCTION

The rapid proliferation of internet-connected devices and cloud computing platforms has created an unprecedented opportunity to modernise healthcare delivery systems. Traditional hospital appointment workflows — characterised by paper registers, telephone scheduling, and manual record maintenance — remain a significant source of operational inefficiency, appointment conflicts, and patient dissatisfaction. These limitations are especially pronounced in resource-constrained environments where administrative capacity is limited, and patient volumes are high.

MediConnect is a production-grade, full-stack Hospital Appointment Management System developed to address these systemic shortcomings. The application is engineered as a single-page web application (SPA) and deployed using the JAMstack paradigm — a modern web architecture combining JavaScript, APIs, and pre-rendered markup.

The system supports two primary user roles: patients, who may register, browse physicians by department, book and track appointments, and review prescriptions and medical history; and administrators, who manage staff records, departmental structures, appointment statuses, and audit logs. A core architectural objective is real-time data consistency — all UI state changes resulting from database mutations propagate to connected clients within sub-second timeframes via WebSocket-based subscriptions.

II. LITERATURE SURVEY

Sr. No	Author(s) & Year	Title	Key Findings	Limitations	Relevance to Proposed System
1	World Health Organisation (WHO), 2011	Strengthening Health Information Systems	Provides a global framework for developing robust health information systems (HIS), including data collection, analysis, and dissemination	Generic guidelines not tailored to specific technologies or implementation environments.	Provides foundational justification for building a structured hospital management system with reliable data flows.

Sr. No	Author(s) & Year	Title	Key Findings	Limitations	Relevance to Proposed System
			strategies.	Lacks technical specifications.	
2	M. A. Khan, S. Ullah, and N. Ahmad, 2015	Design and Implementation of a Hospital Management System Using Web Technologies	Demonstrates a web-based HMS covering patient registration, appointment scheduling, and billing using standard web technologies.	Uses an older web stack; lacks real-time capabilities, mobile responsiveness, and modern API design.	Confirms the feasibility of web-based HMS; motivates the use of modern JAMstack and RESTful API approach in the proposed system.
3	R. Gupta and A. Sharma, 2018	A RESTful API-Based Approach for Hospital Information System Design	Proposes a modular RESTful API architecture for HIS, enabling interoperability between hospital departments and external systems.	Does not address real-time data updates or WebSocket integration; limited discussion on security and authentication.	Directly supports the REST API design used in the proposed system for patient data management and departmental communication.
4	I. Fette and A. Melnikov, 2011	The WebSocket Protocol (IETF RFC 6455)	Defines the WebSocket protocol standard, enabling full-duplex communication channels over a single TCP connection for real-time web applications.	Protocol-level document with no application-specific implementation guidance; requires additional frameworks for practical use.	Provides the technical protocol foundation for real-time features (e.g., live notifications, chat) in the proposed hospital system.
5	Google Firebase, 2025	Firebase Realtime Database Documentation	Firebase Realtime Database offers cloud-hosted NoSQL storage with real-time synchronisation, offline support, and cross-platform SDKs.	Vendor lock-in with the Google ecosystem may incur higher costs at scale, with limited complex querying compared to relational databases.	Evaluated as a real-time backend option; highlights trade-offs that led to choosing Supabase for structured relational data in the proposed system.
6	M. Biilmann, 2016	The JAMstack: A New Front-End Stack for Web Development	Introduces JAMstack architecture (JavaScript, APIs, Markup), emphasising pre-built static files, CDN delivery, and decoupled backend services for performance and security.	Dynamic and real-time features require additional services; not ideal for highly dynamic content without backend augmentation.	Serves as the architectural blueprint for the proposed system, justifying the use of a static frontend with Supabase and Netlify backend services.

III. OBJECTIVES

1) To develop a hospital appointment management system for easy scheduling.
2) To reduce manual work, such as paper records and phone booking.
3) To provide real-time appointment updates using WebSockets.
4) To allow patients to book, view, and track appointments online.
5) To help administrators manage doctors, patients, and records.
6) To ensure secure data access using authentication and role-based control.
7) To create a fast and user-friendly web application using modern technologies.

IV. SYSTEM METHODOLOGY

A. Requirements Analysis

Functional requirements were derived from the operational needs of a mid-scale hospital outpatient department. Core requirements identified include: patient self-registration and authentication; physician and department directory management; appointment booking with real-time availability validation; medical record and prescription access; administrator-level CRUD operations across all system entities.

B. Technology Stack Selection

The technology stack was selected to fulfil all functional and non-functional requirements without traditional server infrastructure. The frontend is implemented as a Vanilla JavaScript SPA, avoiding framework overhead while retaining full SPA routing capabilities. Supabase was selected as the BaaS provider for its PostgreSQL-native relational model, built-in Auth module, granular RLS support, and WebSocket-based Realtime engine — capabilities unavailable in comparable NoSQL alternatives such as Firebase.

C. System Architecture

The system architecture follows a three-tier logical model mapped onto serverless cloud services. The Presentation Tier consists of the pre-built, statically hosted HTML/CSS/JavaScript SPA served via Netlify CDN. The Logic and Data Tier is provided by Supabase, which exposes a PostgREST auto-generated REST API over the PostgreSQL schema, an Authservice for session management.

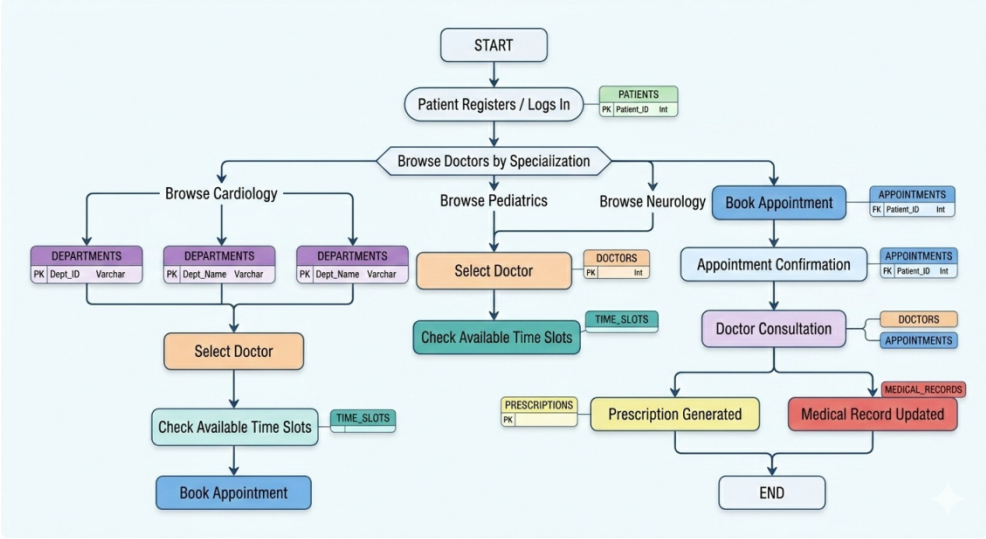


Fig. 1: flowchart of MediConnect

D. Real-Time Engine and Notification Pipeline

Supabase Realtime leverages the PostgreSQL logical replication protocol to stream row-level change events (INSERT, UPDATE, DELETE) over persistent WebSocket connections to subscribed clients. On the client side, subscription handlers update the in-memory application state and re-render the relevant UI components without full-page reloads

E. Development and Deployment Workflow

Development followed an iterative, feature-driven workflow with Git version control. Each feature branch was merged into the main branch upon passing manual functional testing. Netlify's continuous deployment pipeline automatically detects main-branch commits, rebuilds the static site, and propagates the updated build to its CDN edge nodes.

V. MODULE DESIGN

A. Authentication and User Management Module

This module manages user identity lifecycle using Supabase Auth. It handles patient self-registration with email/password credentials, secure sign-in with JWT session token issuance, password reset via email OTP, and session persistence across page reloads. Upon successful registration, a corresponding patient_profile record is automatically created in the PostgreSQL schema via a Supabase database trigger, linking the auth.users UUID to the application-level user record. Administrator accounts are provisioned directly in the database to prevent self-escalation.

B. Patient Portal Module

The Patient Portal is the primary patient-facing interface, presenting a personal dashboard that aggregates upcoming appointments, recent medical records, and prescription summaries. The Specialist Directory sub-component queries the doctors and departments tables to render a browsable physician catalogue with real-time availability indicators sourced from the time_slots table. The Appointment Booking sub-component validates selected time slots against live database state before submitting an INSERT to the appointments table, preventing race-condition double-booking.

C. Administrator Dashboard Module

The Administrator Dashboard provides a comprehensive management console accessible exclusively to administrator-role sessions, enforced at the RLS layer. It comprises five management panels: (1) Doctor Management — CRUD operations on physician profiles and department assignments; (2) Department Management — creation and update of hospital departments; (3) Appointment Management — filterable registry of all system appointments with inline status transition controls; (4) Patient Management — read access to all registered patient profiles and their appointment histories; and (5) Activity Log — a chronological audit trail of administrative actions recorded in the activity_logs table.

D. Real-Time Notification Module

This module establishes and manages WebSocket subscriptions to Supabase Realtime channels scoped to the appointments table. On receiving a change event, the module dispatches an update action to the client-side state store, which triggers a targeted re-render of the affected UI region — the appointment list or availability indicator — without disrupting unrelated interface components. Subscription scope is filtered by the authenticated user's UUID for patient sessions and unfiltered for administrator sessions, aligning with the RLS access model.

E. Deployment and Hosting Module

The Deployment Module encompasses the Netlify-based static hosting and CI/CD pipeline. The production build consists of the pre-rendered HTML entry point, CSS stylesheet, and the unminified JavaScript SPA bundle (total payload < 200 KB). Netlify's build hooks are configured to trigger on every push to the main Git branch, ensuring continuous delivery. HTTPS is enforced via an automatically provisioned TLS certificate. Environment variables for Supabase project URL, anon key, and EmailJS service identifiers are stored as Netlify environment variables, keeping credentials out of the source repository.

VI. LIMITATIONS & FUTURE WORK

While MediConnect demonstrates a complete and deployable implementation, several avenues for enhancement have been identified:

1) Framework Migration: Refactoring the frontend to a component-based framework such as React or Vue.js would improve maintainability, enable code splitting, and facilitate the adoption of a design system library for consistency at scale.
2) Payment Gateway Integration: Incorporating a payment processor (e.g., Razorpay, Stripe) would enable online consultation fee collection, extending the platform towards full revenue-cycle support.
3) Telehealth Module: Integration with a WebRTC-based video conferencing API would enable in-browser teleconsultations, extending utility beyond appointment scheduling to remote care delivery.
4) Stricter RLS Hardening: The current RLS configuration covers core access patterns; a security audit to identify edge cases — particularly for administrative bulk operations — is recommended before large-scale production deployment.

5) Mobile Application: A companion mobile application built with React Native or Flutter, sharing the same Supabase backend, would increase accessibility for patients in mobile-first usage environments common in India.

6) Analytics Dashboard: Embedding aggregated reporting views covering appointment volume trends, physician utilisation, and cancellation rates would provide operational intelligence capabilities for hospital management.

VII. CONCLUSION

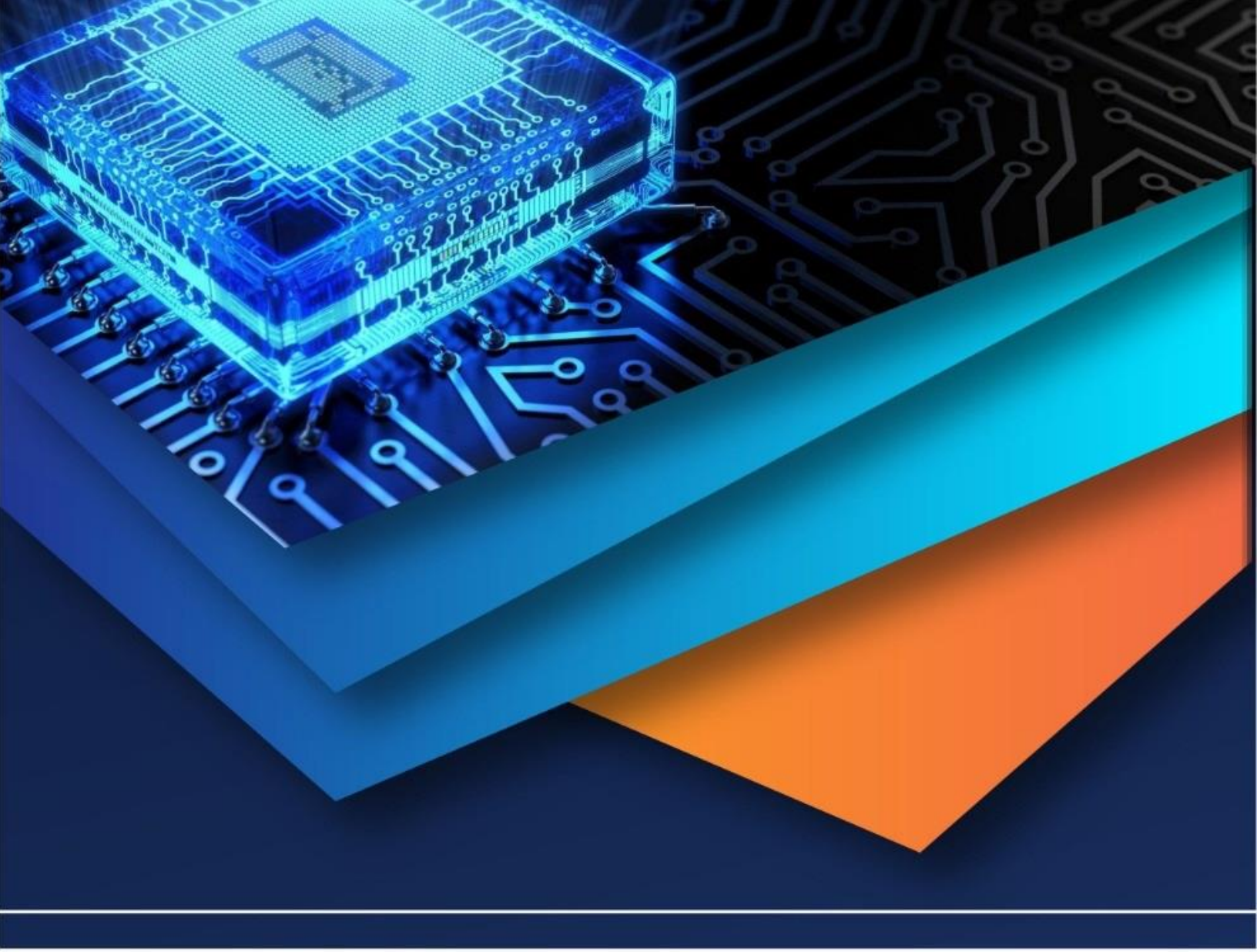
MediConnect successfully demonstrates the design and implementation of a comprehensive, end-to-end hospital appointment management system constructed entirely from modern web technologies and cloud-based SaaS platforms. By adopting a serverless, JAMstack architecture, the project delivers a production-deployable, feature-complete application requiring no traditional server infrastructure to operate or maintain.

The integration of Supabase, Netlify, and EmailJS validates the efficacy of the JAMstack approach for healthcare applications: complex capabilities — including real-time data synchronisation via WebSockets, secure role-based access control enforced through database-level RLS policies, and automated transactional email communications — are achieved through well-composed third-party services rather than bespoke server code. The relational schema accurately models the structural complexity of hospital operations across eight interconnected entities, while WebSocket-based real-time subscriptions maintain consistent application state across concurrent user sessions with sub-second latency.

MediConnect serves as a scalable, extensible foundation for the broader digitisation of healthcare workflows, demonstrating core competencies in full-stack web engineering: advanced relational database design, secure user authentication, event-driven system architecture, and production deployment practices. The system offers a seamless and professional experience for both patients and medical administrators alike, and its serverless nature makes it immediately accessible to healthcare providers at any scale.

REFERENCES

[1] World Health Organisation, "Strengthening Health Information Systems," WHO Press, Geneva, Switzerland, 2011.
[2] M. A. Khan, S. Ullah, and N. Ahmad, "Design and Implementation of a Hospital Management System Using Web Technologies," International Journal of Computer Science and Network Security, vol. 15, no. 3, pp. 88–94, 2015.
[3] R. Gupta and A. Sharma, "A RESTful API-Based Approach for Hospital Information System Design," Procedia Computer Science, vol. 125, pp. 700–706, 2018.
[4] I. Fette and A. Melnikov, "The WebSocket Protocol," IETF RFC 6455, December 2011. [Online]. Available: https://tools.ietf.org/html/rfc6455
[5] Google Firebase, "Firebase Realtime Database Documentation," [Online]. Available: https://firebase.google.com/docs/database, Accessed: 2025.
[6] M. Biilmann, "The JAMstack: A New Front-End Stack for Web Development," Smashing Magazine, 2016. [Online]. Available: https://www.smashingmagazine.com



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY