© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715431

Smart Transit: A Web-Based Public Bus Management System with Real-Time Tracking, QR Ticketing, and Crowd Analytics

J BALA MUKESH[1], DR S PARTHASARATHY[2]
[1] *PG Student, Department of Computer Applications, SRM Valliammai Engineering College, Kattankulathur, Chennai.*
[2] *Professor & Head, Department of Computer Applications, SRM Valliammai Engineering College, Kattankulathur. Chennai.*

Abstract*—Urban public bus transportation in most Indian cities operates without real-time vehicle visibility, relying on manual fare collection and static schedules. This paper presents Smart Transit, a browser-based transit management platform that addresses these gaps through four integrated subsystems: a role-based web application serving Admin, Driver, Conductor, and Passenger actors; a Supabase-backed real-time data pipeline using WebSocket broadcasts; a QR ticket issuance and conductor validation workflow; and a geofence-driven automatic passenger drop-off mechanism. The frontend is built with React 18 on Vite; the backend uses Supabase for PostgreSQL persistence, JWT authentication, and row-level security enforcement. Bus movement is maintained through either native GPS acquisition or a deterministic simulation engine that advances the vehicle along pre-configured stops at 30-second intervals, with linear interpolation producing smooth map animation via Leaflet.js. Passengers book seats by selecting source and destination stops; a UUID-derived QR token is issued for conductor scanning. Proximity to the destination stop within 30 metres—computed via the Haversine formula—triggers automatic disembarkation without conductor interaction. Firebase Cloud Messaging delivers push alerts for bus arrival and destination events. An administrative analytics dashboard aggregates ticketing volume, revenue, and real-time occupancy through PostgreSQL RPC queries. The system demonstrates a practical, low-infrastructure approach to intelligent transit management suitable for mid-scale urban bus networks.

***Keywords*— Smart Transit; Real-Time Bus Tracking; Proximity Detection; Websocket; Haversine Formula**

I. INTRODUCTION

Public bus transportation in most Indian cities still depends on paper tickets, static timetables, and no mechanism for passengers to know where their bus currently is. Route administrators have limited aggregated data to guide decisions about frequency adjustment or fleet allocation. The result is predictable: bunching at stops during peak hours, revenue leakage from undocumented boardings, and passenger frustration from no-show buses.

A unified digital platform covering the complete trip cycle—from route configuration through automatic disembarkation—and making live operational state visible to all parties simultaneously would address most of these inefficiencies. The challenge is doing so without requiring specialised hardware in every vehicle or at every stop, which has historically made such deployments expensive to pilot in academic settings.

Smart Transit takes a browser-first approach. The entire system runs on standard web browsers and commodity smartphones. Real-time state is managed through Supabase's WebSocket broadcast layer, eliminating the need for a custom application server. A simulation engine built into the Driver dashboard allows full end-to-end operation without GPS-equipped vehicles, making the system immediately evaluable in academic and pilot contexts.

II. LITERATURE REVIEW

A. Real-Time Transit Information

WebSocket-based architectures, standardised in RFC 6455, subsequently enabled sub-second event delivery to connected clients without polling overhead [2]. Cloud database platforms that expose real-time table subscriptions over WebSocket have since demonstrated comparable responsiveness for small-to-medium deployments at substantially lower infrastructure cost than dedicated message brokers [3].

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715431

B. Electronic Ticketing and QR Validation

QR-based fare collection has been deployed at scale in metro systems across East and South Asia, consistently reducing average boarding times and fare evasion rates compared to paper ticket workflows [4]. Server-side token validation, where the QR payload carries only an opaque identifier resolved against a backend record at scan time, provides strong forgery resistance without requiring the scanning device to hold a local key database [5]. Integrating QR validation with live occupancy tracking has received limited treatment in the academic literature, as most deployments separate ticketing and capacity monitoring across distinct vendor systems.

C. Geofence-Based Passenger Events

Proximity detection using the Haversine great-circle formula has been applied in ride-hailing platforms to trigger pickup and drop-off events without passenger interaction [6]. Research on urban bus GPS accuracy identifies 30 metres as a practical geofence threshold: narrow enough to distinguish adjacent stops, yet wide enough to remain reliable under the 10–20 metre horizontal drift typical of consumer-grade GNSS receivers in built-up areas [7].

D. Push Notification Architectures

Firebase Cloud Messaging has become the dominant notification substrate for web and mobile transit applications because its device-token model works uniformly across Android, iOS, and desktop browsers without per-platform integration overhead [8]. Geofence-triggered notifications, dispatched only when a bus genuinely approaches a relevant stop, produce substantially higher engagement and lower opt-out rates than schedule-based broadcasts [9].

E. Simulation in Transit Research

Frameworks such as SUMO (Simulation of Urban MObility) support large-scale traffic modelling but require extensive topology configuration and cannot be embedded inside a production web application [10]. A deterministic stop-to-stop advance mechanism, as implemented in this work, provides a lightweight alternative sufficient for validating the complete system workflow in academic and pilot contexts without physical vehicles.

III. SYSTEM ARCHITECTURE

Smart Transit is structured as a three-tier web application. The presentation layer is a React 18 Single-Page Application (SPA) built with Vite. The data layer is Supabase, providing PostgreSQL for persistence, JWT-based authentication, and Realtime WebSocket channels for live event propagation. External services—Leaflet.js for map rendering and Firebase Cloud Messaging for push delivery—are integrated via their browser JavaScript SDKs. No custom application server is deployed; business logic not expressible as SQL executes either client-side or inside Supabase database functions.

The architecture follows three distinct tiers: (1) Client Tier — React 18 SPA running in the browser, rendering role-specific dashboards for Admin, Driver, Conductor, and Passenger; (2) Service Tier — Supabase providing PostgreSQL persistence, JWT Auth, Row-Level Security, Realtime WebSocket engine, and Edge Functions; (3) External Services — Leaflet.js for interactive map rendering, Firebase Cloud Messaging for cross-platform push notifications, and a QR generation/scanning library for ticketing.

The architecture achieves real-time delivery through Supabase's table-change broadcast. When the Driver dashboard inserts a row into bus_positions, the Realtime engine immediately pushes the event to all WebSocket clients subscribed to that trip's channel. Passenger dashboards receive the coordinate and feed it into the interpolation engine without any polling.

Fig. 3.1. System architecture

A. Role-Based Access Control

Four roles are defined: Admin, Driver, Conductor, and Passenger. Each role is stored in the users table

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715431

and embedded in the Supabase JWT on authentication.

TABLE 1 Role Capabilities

Role	Key Capabilities
Admin	Route and stop creation, fleet management, driver assignment, analytics dashboard
Driver	Trip initiation, GPS or simulation mode selection, real-time position broadcast
Conductor	QR ticket scanning, passenger validation, occupancy updates
Passenger	Seat booking, QR ticket receipt, FCM arrival and destination alerts, auto drop-off

Frontend route guards redirect unauthenticated or under-privileged users before any data fetch occurs. Supabase row-level security (RLS) policies enforce the same constraints at the database layer, ensuring that a compromised client cannot read or write data outside its permitted scope.

B. Frontend Architecture

Vite's optimized production bundler reduces initial load size. Role-specific routes are wrapped in a PrivateRoute component that reads JWT claims and redirects on mismatch. The Leaflet map is initialized inside a useRef hook to prevent re-instantiation on component re-renders, and bus markers are managed via direct Leaflet API calls outside React's reconciliation cycle to avoid the frame-rate penalty of DOM diffing on every position update.

C. Backend Architecture

Supabase Auth issues RS256-signed JWTs carrying the user's role as a custom claim. PostgreSQL RLS policies evaluate this claim on every query, enforcing that passengers see only their own bookings, conductors update only their assigned trip's records, and drivers write position data only for their active trip. Database functions exposed as RPC endpoints handle aggregate computations for the analytics dashboard, returning pre-joined JSON that minimises round-trips from the frontend.

D. Notification Layer

On first login each passenger client calls the FCM SDK to obtain a device registration token, which is stored in the users table. A Supabase database trigger on bus_positions inserts evaluates whether the new coordinate falls within 30 metres of any boarded passenger's source or destination stop and, if so, invokes an Edge Function that dispatches the corresponding FCM payload to the stored token. Notification dispatch is therefore server-side and delivers even when the passenger's browser tab is inactive.

IV. IMPLEMENTATION METHODOLOGY

The system is implemented as a browser-native web application using the technology stack summarised in Table II. The approach avoids specialised hardware at vehicles or stops, reducing deployment cost and enabling full evaluation in academic settings using the built-in simulation engine.

A. Route and Stop Configuration

The Admin map interface loads a full-screen Leaflet instance. Each map click appends a stop record—with latitude, longitude, and sequence index—to the routes table.

TABLE 2 Technology Stack

Layer	Technology	Purpose
Frontend	React 18 + Vite	SPA, role-based dashboards
Backend	Supabase / PostgreSQL	Database, auth, realtime engine
Maps	Leaflet.js	Route, stop display, bus tracking
Notifications	Firebase Cloud Messaging	Push alerts for passengers
QR	QR generator + scanner lib	Ticket issuance and validation
Proximity	Haversine formula (30 m)	Automatic passenger drop-off
Animation	Linear interpolation	Smooth bus movement on map

The route geometry is rendered as a polyline connecting stops in order. Bus creation, capacity setting, and driver assignment are handled through form components that submit directly to Supabase,

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715431

with validation enforced at the database constraint level rather than client-side logic alone.

B. Driver Dashboard and Trip Initiation

Before starting a trip the driver selects GPS Mode or Simulation Mode. In GPS Mode the browser Geolocation API streams position fixes from device hardware; each fix is written as a new bus_positions row. In Simulation Mode a setInterval callback fires every 30 seconds, reads the coordinates of the next stop in the route array, and writes them as a new position row. Both paths produce identical bus_positions records, making them completely transparent to all downstream consumers including the passenger map and the proximity engine.

C. Bus Position Interpolation

Rendering each received coordinate update as an immediate marker teleport produces a jarring visual at the 30-second simulation interval. Instead, the passenger dashboard stores each received position as a target and runs a request Animation Frame loop that advances the marker along a straight-line path from its current rendered location to the target. The step size is calibrated to complete the transit within the expected inter-update interval, producing fluid animation without requiring knowledge of intermediate road geometry.

D. Passenger Booking

Authenticated passengers select a source stop and destination stop from a dropdown populated with the active route's stop list. The system queries for buses assigned to that route and displays current occupancy. On confirmation the backend inserts a booking record with a UUID v4 QR token and returns it to the client, which renders the QR code using a JavaScript generation library. The token is the only booking identifier exposed in the QR payload; the booking ID accompanies it solely to accelerate the conductor-side database lookup.

E. Database Design

The schema is hosted in Supabase's managed PostgreSQL instance and comprises six tables whose foreign key relationships enforce the operational logic of the transit system at the data layer rather than relying on application-layer checks alone. Table III summarises the key columns of each table.

TABLE 3 Database Schema Summary

Table	Key Columns
users	id (PK), role, name, email, fcm_token
routes	id (PK), name, stops (JSONB array), created_by
buses	id (PK), registration, capacity, route_id (FK), driver_id (FK)
trips	id (PK), bus_id (FK), route_id (FK), driver_id (FK), status, started_at
bookings	id (PK), passenger_id (FK), trip_id (FK), src_stop, dst_stop, qr_token, status
bus_positions	id (PK), trip_id (FK), latitude, longitude, timestamp

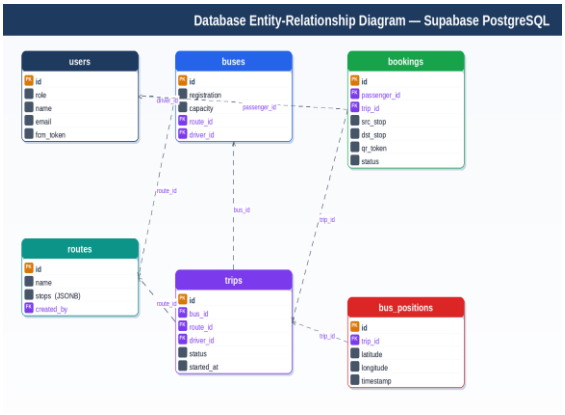

Fig. 4.1. Entity-relationship diagram for the Supabase PostgreSQL schema. Grey lines denote foreign key references; all relationships enforce ON DELETE constraints to prevent orphaned records.

V. SIMULATION AND TRACKING ENGINE

Physical GPS-equipped buses are unavailable in academic development environments, yet a transit management system must be fully testable without them. The simulation engine bridges this gap by reproducing the observable behaviour of a moving bus—a sequence of coordinate updates written at regular intervals—using only a web browser running the Driver dashboard.

A. Simulation Mode

The Driver dashboard maintains a stopIndex state variable initialised to zero when the trip starts. A setInterval callback fires every 30,000 milliseconds, retrieves the pre-stored latitude and longitude of the

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715431

stop at stopIndex from the route's stop array, and writes these coordinates to bus_positions as a new row. The index increments after each write and wraps to zero at the terminal stop, enabling continuous looped operation throughout a demonstration session without driver intervention.

B. GPS Mode

In GPS Mode the navigator.geolocation.watchPosition() call streams position updates from the device's GNSS hardware to the same bus_positions write path used by simulation. The maximumAge parameter is set to 5,000 ms to accept cached fixes only within that window; the timeout is set to 10,000 ms. Both modes are mutually exclusive and are locked at trip initiation—switching mid-trip would introduce a discontinuous jump in the recorded path and is blocked by the UI.

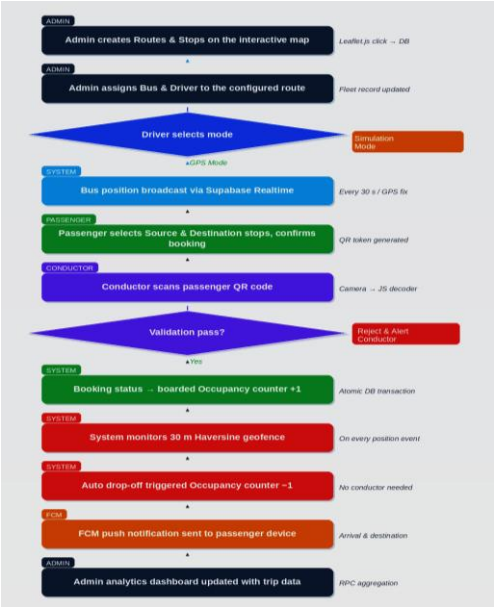

Fig. 5.1. End-to-end system workflow from Admin route configuration through automatic passenger drop-off and analytics dashboard update.

C. Realtime Propagation

Passenger dashboard components subscribe to the bus_positions channel filtered by the active trip_id on mount. Each INSERT event delivered by Supabase Realtime carries the new latitude and longitude as the event payload. The subscription is torn down on component unmount via the Supabase channel removal API, preventing accumulation of stale listeners across navigation events in the SPA.

VI. RESULTS AND DISCUSSION

The Smart Transit system was evaluated through end-to-end functional testing across all four role-based dashboards.

A. Admin Dashboard – Route and Fleet Management

The Admin dashboard enables creation of routes by clicking stops on an interactive Leaflet map. Once routes are defined, buses are assigned to routes and drivers are linked to buses. The analytics panel displays real-time occupancy, revenue, and booking status per route.

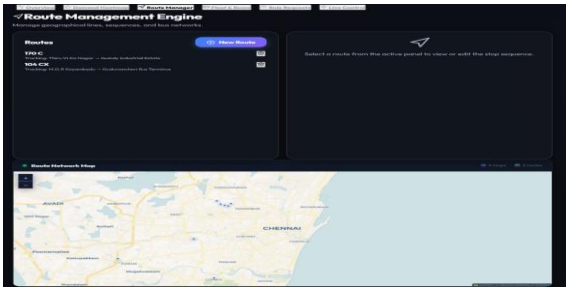

Fig. 6.1. Admin dashboard showing interactive Leaflet map with route stops and bus assignment panel.

B. Driver Dashboard – GPS and Simulation Mode

The Driver dashboard allows the driver to initiate a trip in either GPS Mode or Simulation Mode. In Simulation Mode the bus advances to the next stop every 30 seconds automatically, enabling full workflow demonstration without a physical vehicle. The live position is broadcast via Supabase Realtime immediately upon each update.

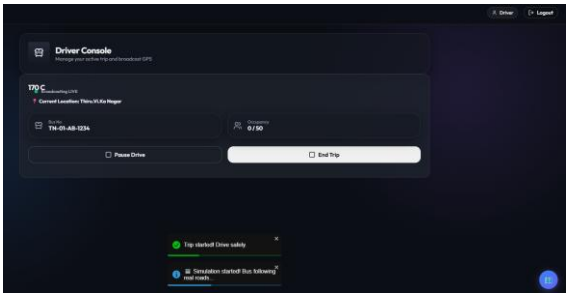

Fig. 6.2. Driver dashboard showing active simulation mode with current stop index and next stop details.

C. Passenger Dashboard – Booking and Live Tracking

Passengers select source and destination stops from a dropdown populated with the active route's stop

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715431

list. Upon confirmation, a QR code is rendered on screen. The map view displays the live bus position updated in real time via WebSocket, with smooth linear interpolation between updates.

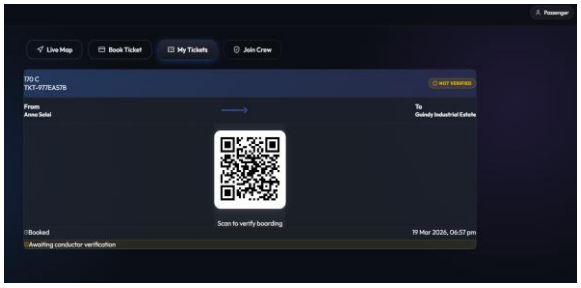

Fig. 6.3. Passenger dashboard showing booking confirmation with QR code and current seat availability.

TABLE IV Feature Comparison

Feature	Proposed	GTFS App	Static QR App	GPS Tracker
Real-time bus tracking	Yes	Yes	No	Yes
QR ticket issuance	Yes	No	Yes	No
Conductor QR validation	Yes	No	Partial	No
Auto drop-off (30 m)	Yes	No	No	No
Simulation mode	Yes	No	No	No
FCM push notifications	Yes	Partial	No	No
Real-time occupancy	Yes	No	No	No
Admin analytics dashboard	Yes	No	Partial	No
Role-based access control	Yes	No	Partial	No

XII. LIMITATIONS AND CHALLENGES

A. Simulation Fidelity

The simulation engine advances to discrete stop coordinates at a fixed interval without modelling inter-stop transit time, dwell time, traffic variability, or road geometry. Passengers viewing the map see linear interpolation paths that may cross buildings or water bodies between stops. Integration with a routing API such as OSRM would constrain movement to the road network at the cost of an external dependency and added latency on each step.

B. GPS Accuracy in Urban Environments

The 30-metre geofence remains susceptible to the 10–20 metre horizontal error common in dense urban environments, particularly near tall buildings where multipath GNSS reflection degrades accuracy. In adverse conditions the geofence may fire one stop early or fail to fire precisely at the correct location. Map-matching or differential correction would reduce this error but are not implemented in the current version.

C. Offline QR Validation

Conductor scanning requires a live connection to the Supabase backend. In areas with intermittent mobile data coverage the validation request may timeout. An offline fallback using an HMAC signature embedded in the QR payload would allow local verification, but would require secure key management on conductor devices and is not yet implemented.

D. Scalability at Large Concurrency

Supabase Realtime currently broadcasts all bus_positions inserts to all connected clients, relying on client-side filtering by trip_id. At large scale—hundreds of concurrent trips—this produces unnecessary bandwidth on each client. Supabase's broadcast channel mode with per-trip topic segmentation would address this but requires additional channel management logic not yet implemented in this codebase.

VII. CONCLUSION

This paper presented Smart Transit, a web-based transit management system that integrates real-time vehicle tracking, QR ticket issuance, conductor validation, proximity-triggered automatic drop-off, push notification delivery, and administrative analytics within a single React and Supabase application. The role-based architecture accommodates four distinct operational actors through purpose-built dashboards that share a

common real-time data substrate without a custom application server.

The simulation engine removes the physical vehicle dependency that has historically blocked academic development of transit systems. Linear interpolation of bus positions between update intervals produces smooth map animation independent of GPS or simulation update frequency. The Haversine geofence-based automatic drop-off eliminates the dependency on manual conductor confirmation of disembarkation, directly improving occupancy data accuracy for the analytics dashboard.

Future work will prioritise: road-network-constrained simulation interpolation via OSRM; offline QR validation using HMAC-signed ticket payloads; dynamic fare calculation based on source-to-destination stop distance; per-trip Supabase Realtime topic segmentation for scalability beyond single-route pilots; and native mobile application wrappers for reliable FCM foreground delivery on iOS and Android.

VIII. ACKNOWLEDGMENT

The author thanks Dr. S. Parthasarathy (HOD/MCA, Department of Computer Applications, SRM Valliammai Engineering College) for guidance and mentorship throughout this project. The author also acknowledges the open-source communities behind Supabase, React, Leaflet.js, and Firebase for the documentation and tooling that made this work possible.

REFERENCES

[1] A. Cottrill and P. Thakuriah, "Evaluating transit information systems for use in transit rider decision making," Transportation Research Record, vol. 2143, pp. 56–64, 2010.
[2] I. Fette and A. Melnikov, "The WebSocket Protocol," RFC 6455, Internet Engineering Task Force, Dec. 2011.
[3] V. Bhatt and N. Sharma, "Cloud-based real-time transit tracking using serverless architectures," in Proc. Int. Conf. Cloud Comput. Emerging Mkts., 2022, pp. 1–6.
[4] C. Pelletier, M.-A. Trépanier, and C. Morency, "Smart card data use in public transit: A literature review," Transportation Research Part C, vol. 19, no. 4, pp. 557–568, 2011.
[5] L. Cheung, "QR code-based authentication and ticketing in mobile transit systems," in Proc. IEEE Int. Symp. Technology and Society, 2019, pp. 1–5.
[6] B. Ferreira, "Geofencing in ride-hailing services: Architecture and accuracy considerations," IEEE Pervasive Computing, vol. 18, no. 3, pp. 44–52, 2019.