

Design and Implementation of a Metro Station Management System

¹Saloni, ²Shreya Singh, ³Ms. Shama Bano

Students, Computer Science And Engineering^{1,2}

Assistant Professor, Computer Science And Engineering³

Kamla Nehru Institute of Physical and Social Sciences, Sultanpur, India

Abstract: *A modern metro station is a complex environment in which ticketing, passenger movement, train information, maintenance, and security have to work together. In practice, these activities are often handled through separate systems, which can make information sharing slow and increase the amount of manual coordination required from station staff. This paper presents a Metro Station Management System that brings six operational areas together on a common platform: QR-based ticketing with distance-based fares, passenger-flow monitoring, train scheduling and live arrival information, staff and maintenance management, CCTV-linked security alerts, and an administrative dashboard. The proposed system uses React and Node.js for the web application, MySQL for persistent data, and IoT-based sensors for occupancy monitoring. A relational database was selected to maintain consistency among ticket, fare, entry, and audit records. The design focuses on three critical operational paths—ticket validation, duplicate-scan prevention, and alert delivery—with target response times of less than one second for gate validation and less than five seconds for dashboard alerts. These targets are design objectives rather than physical-hardware measurements at this stage. The paper also discusses the current limitations, particularly sensor accuracy, network dependence, and the prototype's single-station scope.*

Keywords: metro station management, QR ticketing, crowd monitoring, IoT sensors, train scheduling, real-time dashboard.

I. INTRODUCTION

Metro systems transport very large numbers of passengers every day, and the station is the point where many operational activities come together. Delhi Metro reported an average of about 6.46 million passenger journeys per day in 2025, with a single-day peak of more than 8.18 million on 8 August 2025 [1], [2]. Behind every journey, station staff continuously make small but important decisions: whether a gate is functioning, whether a platform is becoming crowded, when the next train is expected, and whether a reported fault or security incident requires attention. When these decisions depend on separate screens, registers, and verbal communication, responding quickly becomes more difficult.

The main challenge is therefore not the absence of individual technologies, but the lack of coordination between them. A ticketing system can record how many passengers have entered without knowing the current platform occupancy. A CCTV control room can identify a growing crowd but may not have the train schedule or maintenance status immediately available. Similarly, a maintenance request recorded on paper or in a separate spreadsheet may not reach the station controller until someone

This project aims to address that problem by treating the metro station as one connected operation rather than as a collection of independent systems. The proposed platform integrates six major areas: ticketing and fare calculation, passenger-flow monitoring, train scheduling and arrival information, staff and maintenance management, security alerts, and administrative monitoring. The central idea is simple: information generated in one part of the station should



be useful to the people responsible for another. For example, a rising platform occupancy level can immediately become visible to the controller, while a maintenance issue can be displayed alongside the affected station asset.

Several practical considerations influenced the design. QR tickets were selected instead of dedicated smart-card hardware because they can be issued through a mobile interface and generally require less specialized deployment infrastructure. However, they also require dependable scanning and protection against duplicate use. MySQL was chosen because ticket, fare, staff, and operational records have strong relationships and benefit from transactional consistency. The trade-off is that very high-frequency sensor data may eventually require a specialized storage solution when the system is expanded to many stations.

The remainder of this paper is organized as follows. Section II discusses related work and the standards that influenced the design. Section III defines the operational problems and project objectives. Section IV presents the proposed architecture and modules. Sections V and VI describe the methodology and implementation. Section VII discusses the expected performance and design-based evaluation, while Sections VIII and IX present the limitations and future scope. Section X concludes the paper.

II. LITERATURE REVIEW

Existing work relevant to metro station management can broadly be grouped into four areas: digital ticketing, pedestrian and crowd-flow analysis, public-transit operations, and software communication technologies. Rather than treating these areas independently, this project uses them as building blocks for a single station-level management platform. Established standards and research references were used wherever they directly supported a design decision.

QR codes are standardized by ISO/IEC 18004 [3] and include error-correction mechanisms that help a code remain readable even when a printed or displayed code is partially damaged. This is useful in a metro environment, where a passenger may present a dim or scratched phone screen at a gate. The QR standard itself, however, does not determine whether a ticket is valid, expired, or already used. Those security and business rules must be implemented by the ticketing application. A design-science study involving 176 Bangkok BTS Skytrain passengers found that conventional card tickets were still preferred by many participants, while trust and efficiency strongly influenced acceptance of QR ticketing [4]. That finding highlights an important point: a QR system must be convenient as well as reliable.

For crowd management, Fruin's pedestrian level-of-service framework [5] provides a way to describe pedestrian and queue density from free movement (A) to severe breakdown (F). The Transit Capacity and Quality of Service Manual [6] also provides capacity guidance for platforms, stairs, and fare-control areas. This project does not attempt to replace those established frameworks. Instead, it adopts the general idea of graded crowd levels and applies configurable thresholds to platform occupancy. Research on IoT-based crowd sensing in public transportation similarly describes systems that monitor and predict crowding and use that information to support operational decisions [7]. Our implementation represents a narrower application of this concept: count passengers, compare the count with platform capacity, and raise an alert when a threshold is crossed.

The software architecture also relies on established technologies. A relational data model [8] is appropriate for information such as fares and tickets, where consistency is important. REST [9] is used for application programming interfaces, WebSocket [10] provides real-time updates to connected dashboards, MQTT [11] transports sensor messages, and JSON Web Tokens [12] support staff authentication. None of these technologies is novel individually. The contribution of the proposed system is instead their integration at the station level. The sources reviewed address individual areas such as QR-ticket adoption [4] or crowd sensing [7], whereas this design combines ticketing, passenger flow, train information, maintenance, and security in a shared operational view. This observation is limited to the sources reviewed for this project and is not presented as a complete survey of the field.

III. PROBLEM STATEMENT

The project was motivated by three closely related operational problems. First, station information is fragmented: entry counts may exist in the ticketing system, camera information may remain inside the control room, and maintenance



requests may be stored in paper records or informal communication channels. As a result, there is no single operational view of the station. Second, crowding may be recognized only after it has already become a problem. A controller may have to depend on a camera feed or a report from a guard before taking action. Third, staff and maintenance activities are often recorded manually, making shift changes, fault reports, and repair progress more difficult to track and audit. The objective of the proposed system is therefore to provide one platform for ticket issuance and validation, platform occupancy monitoring, train-arrival information, staff assignment, maintenance tracking, security and safety alerts, and administrative oversight. Two measurable design targets were defined: gate-level ticket validation should complete in less than one second, and an alert should become visible on the dashboard within five seconds of the triggering event. These values represent practical design goals rather than confirmed hardware results and are evaluated accordingly in Section VII.

IV. PROPOSED SYSTEM

A. Architecture

The proposed system is organized into four main layers. Field devices—including gate scanners, platform occupancy sensors, and camera-based alert sources—send events through MQTT to a station message broker. A Node.js backend receives these events, stores relevant information in MySQL, and exposes a REST API for application operations. The same backend uses WebSocket connections to push live changes to the React front end. Access is role-based, so gate staff, controllers, maintenance supervisors, and administrators see the information relevant to their responsibilities. A message broker is placed between the field devices and the application backend because sensors and scanners may temporarily lose connectivity or reconnect. The broker can buffer messages during short interruptions, reducing the need for the backend to handle every device-level connection directly. This improves resilience, although it also introduces an additional service that must be configured, monitored, and maintained.

B. Modules

Ticketing and fare calculation. A passenger selects an origin and destination, after which the server calculates the fare using station-pair distance information and a configured fare slab. A unique ticket identifier is generated and returned with a signed QR payload. At entry, the ticket is marked as being used, and at exit the journey is closed and the fare status is confirmed.

Passenger flow and crowd monitoring. Counters installed near platform entrances report passenger entries and exits. The backend maintains a running occupancy count for each platform and compares it with the configured capacity. When the occupancy crosses a warning or critical threshold, the corresponding status is made visible to the control dashboard.

Train scheduling and arrival display. Timetables are maintained for each line and direction. When a controller records a delay, the system recalculates the expected arrival information and updates the connected display interfaces without requiring a manual page refresh.

Staff and maintenance management. Supervisors can create shifts and assign employees to specific posts. Staff members can also create maintenance tickets against station assets such as escalators, lifts, or gates. Each ticket moves through defined states—from open, to assigned, to resolved—with timestamps that provide a basic audit trail.

Security and CCTV alerts. Alerts generated by camera analytics or entered manually are associated with a camera identifier and station location. Each alert receives a severity level, and important alerts require acknowledgement by an identified staff member. This makes it easier for controllers to distinguish routine events from incidents that need immediate attention.

Administrative dashboard. The dashboard brings the most important operational information into a single view, including platform occupancy, upcoming trains, unresolved maintenance tickets, active security alerts, and daily ticket activity. The intention is to reduce the need for staff to switch repeatedly between different screens during busy periods.



V. METHODOLOGY

An iterative development approach was followed. The data model was defined first, followed by the implementation and testing of individual modules from end to end. After a module was validated, the next module was integrated with the existing system. The requirements were informed by the standards and studies discussed in Section II and by publicly available reporting on the operation of large metro systems such as Delhi Metro [1], [2]. Direct interviews with station staff were not conducted, although such interviews would be valuable in a future evaluation.

Fare calculation. In the proposed model, stations are represented as nodes on a line, and the distance between neighboring stations is stored in kilometres. The distance between station pairs is calculated in advance and stored in a station-pair table, allowing the system to avoid repeating route calculations for every ticket purchase. The resulting distance is mapped to a fare slab. The prototype uses illustrative slabs—for example, up to 2 km at ₹10, 2–5 km at ₹20, and 5–12 km at ₹30—rather than claiming to represent an actual operator's fare policy. Precomputing the values makes fare lookup fast and easier to audit, although a change in the network would require the station-pair data to be regenerated. **Ticket validation.** Each QR payload contains the ticket identifier, origin, destination, expiry information, and an HMAC signature. When a scanner submits the code, the backend verifies the signature, checks whether the ticket has expired, and confirms its current status. Duplicate use is treated as a critical case. To prevent two copies of the same ticket from being accepted at nearly the same time, the database performs a conditional update that changes the ticket from ISSUED to IN_USE only when the original status is still ISSUED. The database therefore determines which concurrent request succeeds, avoiding the need for separate application-level locking.

Crowd thresholds. Platform occupancy is calculated as the difference between recorded entries and exits, compared with the platform's configured design capacity. The prototype uses 60% as a warning threshold and 80% as a critical threshold. These values are configurable starting points inspired by the graded-density approach discussed in [5]; they are not values prescribed by a standard and should be adjusted after observing the characteristics of each platform.

Alert handling. Every alert is assigned a severity level from 1 to 3. Level-3 alerts are placed prominently on the dashboard, accompanied by an audible notification, and remain visible until a responsible user acknowledges them. This acknowledgement step is intentional: an alert that disappears automatically can easily be overlooked during a busy operational period.

VI. IMPLEMENTATION

Requirement	Target	Design basis
Gate validation time	<1s	One network round trip, one signature check, and one indexed update
Duplicate scan rejected	100%	Conditional update changes one row or none; database locking serializes concurrent scans
Fare matches fare table	100%	Single lookup using precomputed station-pair distances and fare slabs
Alert latency, sensor to dashboard	<5s	One MQTT publish, one database insert, and one WebSocket push; no polling
Dashboard refresh	<2s	Dashboard updates are pushed when data changes rather than on a fixed timer

The front end is implemented in React with role-based routes, while the backend uses Node.js and Express. REST APIs handle operations such as ticket purchases and shift creation, whereas WebSocket connections push occupancy changes, train-arrival updates, and alerts to active dashboards. MySQL stores persistent application data. Sensors and scanners communicate through a Mosquitto MQTT broker. During prototype development, sensor input is generated by a script that imitates the messages a physical counter could produce—for example, an ESP32-based device using an infrared beam-break sensor. This approach allows the software pipeline to be tested before physical hardware is integrated, but it also means that real-world sensor accuracy has not yet been measured.



The database contains approximately a dozen core tables, including users, stations, station_distances, fare_slabs, tickets, trains, schedules, platforms, sensor_readings, staff_shifts, maintenance_tickets, and alerts. Ticket and fare records are normalized and connected through foreign keys because consistency is more important for these records than marginal write speed. Sensor readings are stored as append-only records indexed by platform and timestamp. A periodic process can summarize older readings into hourly aggregates so that the raw sensor table does not grow indefinitely.

Authentication is handled using short-lived JSON Web Tokens [12], and API endpoints verify the role associated with each authenticated user. Passwords are stored as bcrypt hashes rather than plain text. QR ticket payloads are signed on the server, and the signing key remains on the backend. As a result, a scanner can forward a ticket payload for verification but cannot independently generate a valid forged signature.

The dashboard was deliberately designed around a single operational screen. During a rush period, a controller should be able to identify the most important information—crowding, upcoming trains, and active alerts—without navigating through several unrelated pages. This choice may reduce opportunities for visual decoration, but it prioritizes situational awareness and speed of use.

VII. RESULTS AND DISCUSSION

The proposed system has not yet been evaluated on physical station hardware. Rather than presenting unmeasured performance figures as experimental results, this section evaluates the critical paths against the design targets and explains why the architecture is expected to support them. The hardware evaluation should ultimately confirm whether the targets hold under realistic network conditions and passenger loads.

TABLE I. TARGETS AND THE DESIGN BASIS FOR MEETING THEM

Gate validation. A typical validation request involves one network round trip, a signature verification step, and an indexed database update. Each operation is relatively small, so achieving a sub-one-second response on a local station network appears technically reasonable. In a real deployment, network latency is likely to be one of the main contributors to response time, which makes the cached validation fallback discussed in the limitations section important.

Duplicate scans. The conditional database update either changes one ticket record or changes nothing. InnoDB provides row-level locking for the update, so simultaneous attempts to use the same ticket are serialized: the first request changes the status, while the second sees that the ticket is no longer ISSUED and is rejected. A useful hardware and load test would therefore send many concurrent scans for a single ticket and verify that exactly one request succeeds.

Alert delivery. The alert path consists of an MQTT publication, a database insertion, and a WebSocket push to connected dashboards. Because the dashboard is updated through push notifications rather than periodic polling, the delay is not dependent on a refresh interval. In practice, the observed response will still depend on network conditions and broker performance.

When physical testing is conducted, performance should be reported using both median and 95th-percentile latency rather than relying only on averages. This is especially important at a metro gate: a system that is fast most of the time but becomes slow for a small percentage of scans can still create visible queues during peak demand.

VIII. LIMITATIONS

The current evaluation is analytical rather than a measurement on physical hardware, so the timing targets remain unconfirmed. The prototype also represents only a single station, and its behaviour across a network of many stations has not been evaluated. The occupancy model can gradually drift because a missed entry or exit is not automatically corrected; periodic reconciliation with another counting source, such as camera-based counts, would be necessary in a production system. Sensors may also undercount or overcount when passengers move side by side or carry large objects.



The system currently relies on timetable and delay information entered by authorized controllers. It does not connect directly to railway signalling or train-control infrastructure, which are safety-critical systems and would require a separate level of certification and integration. Similarly, the security module receives alerts from upstream sources but does not perform video analysis itself, so its effectiveness depends on the quality of those sources. QR validation also requires network connectivity or a cached set of valid tickets; the prototype does not yet implement a complete long-outage degraded mode.

MySQL is suitable for the scale of the current prototype, but a deployment involving high-frequency sensor readings from many stations would likely benefit from a dedicated time-series database. In addition, camera and passenger-related information introduces privacy and data-governance considerations that would need to be addressed according to the laws and policies applicable to the deployment environment.

IX. FUTURE SCOPE

The most important next step is to extend the prototype from a single station to a multi-station architecture with a centralized control view. With sufficient historical occupancy data, the system could move beyond simple threshold alerts and forecast crowding by time of day, allowing operators to issue announcements or take preventive action before a platform becomes congested. A passenger-facing mobile application could provide ticket purchasing and live train information before passengers reach the station. Support for card and UPI payments could complement QR tickets. For security, on-device video analytics could reduce reliance on manual alert triggers, while a time-series database could provide more efficient storage for large sensor streams. Accessibility features, including lift-status information and assistance requests, would also make the platform more useful to passengers with different mobility needs.

X. CONCLUSION

This paper presented a Metro Station Management System that brings ticketing, passenger-flow monitoring, train information, staff and maintenance management, security alerts, and administrative monitoring into a common platform. The design combines a message broker for communication with field devices, a relational database for consistent operational records, a conditional database update for preventing duplicate ticket use, and a single-screen dashboard for real-time situational awareness. The architectural analysis indicates that the proposed system can reasonably target sub-one-second ticket validation and sub-five-second alert visibility, but these targets still require confirmation through physical-hardware testing. The current prototype is limited to one station and depends on the reliability of its sensors and network. Even with these limitations, the project demonstrates how a shared data layer can reduce fragmented information and make station operations easier to monitor and coordinate.

REFERENCES

- [1] P. Jain, "Delhi Metro crosses record 235 crore passenger journeys in 2025," United News of India, New Delhi, India, Mar. 15, 2026. [Online]. Available:
- [2] IANS, "Delhi Metro records highest-ever daily ridership on Aug 8," IANS, New Delhi, India, Aug. 9, 2025. [Online]. Available:
- [3] ISO/IEC 18004:2015, "Information technology—Automatic identification and data capture techniques—QR Code bar code symbology specification," Int. Org. Standardization, Geneva, Switzerland, 2015.
- [4] Z. Xie and L. Jonjoubson, "Intention of BTS Skytrain passengers to use QR ticket for metro mass transit system in Bangkok," in Proc. 10th Nat. Int. Conf. Research to Serve Society, Huachiew Chalermprakiet Univ., Samut Prakan, Thailand, Jun. 2023, pp. 163–177.
- [5] J. J. Fruin, *Pedestrian Planning and Design*. New York, NY, USA: Metropolitan Association of Urban Designers and Environmental Planners, 1971.



- [6] Transportation Research Board, Transit Capacity and Quality of Service Manual, 3rd ed., TCRP Report 165. Washington, DC, USA: Nat. Acad. Press, 2013.
- [7] D. Darsena, G. Gelli, I. Iudice, and F. Verde, "Sensing technologies for crowd management, adaptation, and information dissemination in public transportation systems: A review," IEEE Sensors J., vol. 23, no. 1, 2023, doi: 10.1109/JSEN.2022.3223297.
- [8] E. F. Codd, "A relational model of data for large shared data banks," Commun. ACM, vol. 13, no. 6, pp. 377–387, Jun. 1970.
- [9] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. California, Irvine, CA, USA, 2000.
- [10] I. Fette and A. Melnikov, "The WebSocket protocol," IETF, RFC 6455, Dec. 2011.
- [11] A. Banks and R. Gupta, Eds., "MQTT Version 3.1.1," OASIS Standard, Oct. 2014.
- [12] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," IETF, RFC 7519, May 2015.

