International Journal of Creative and Open Research in Engineering and Management
ISSN: **3108-1754 (Online)**
Volume 02 Issue 06 June-2026 | Impact Factor: 3.5

Public Vehicle Notification System Using Android App

Akshaya S

Nagarjuna College of Engineering and Technology
Beedaganahalli, Venkatagiri Kote, Devanahalli, Bengaluru, Karnataka 562110
akshayareddy4947@gmail.com

Roshitha S N
Nagarjuna College of Engineering and Technology
Beedaganahalli, Venkatagiri Kote, Devanahalli, Bengaluru, Karnataka 562110
roshithasn@gmail.com

Sneha B Gudi
Nagarjuna College of Engineering and Technology
Beedaganahalli, Venkatagiri Kote, Devanahalli, Bengaluru, Karnataka562110
gudisneha67@gmail.com

Soumya G
Nagarjuna College of Engineering and Technology
Beedaganahalli, Venkatagiri Kote, Devanahalli, Bengaluru, Karnataka 562110
soumyak6263@gmail.com

Gopinath A R
Nagarjuna College of Engineering and Technology
Beedaganahalli, Venkatagiri Kote, Devanahalli, Bengaluru, Karnataka562110
ar.gopinath@ncetmail.com

Abstract - There are many cars parked at the wrong places in busy cities. This causes problems like blocking of roads, traffic jams, and delays for emergency vehicles. People often aren't aware of whom to contact when a car call the owner and ask them to move their car. It is neither safe nor easy to share phone numbers. We, therefore, It made a mobile app called the Public Vehicle Notification System. Each car has a QR code on it. If a problem is spotted by someone, they can scan the code with the app and send a message to the car owner. The owner immediately receives a notification on their phone. No phone numbers are exchanged and everything is private, while the application also entails safety measures that can prevent its abuse. It can be used by police in Emergencies too. The system makes parking better, keeps the roads clear, and improves communication between people . The Public Vehicle Notification System is an Android-based mobile application designed to address the growing problem of improper parking in busy urban environments. Vehicles parked in restricted or inconvenient location might cause disruption of traffic flow, congestion, and delays. emergency services, such as ambulances and fire trucks. Traditional methods like honking, Announcements or towing are either ineffective or time-consuming. This system introduces a Smarter alternative would enable users to inform vehicle owners instantly using QR codes without revealing any personal contact information.

Keywords- Improper Parking, QR Code, Android App, Traffic Management, Real-time Notification, Privacy Protection, Secure Communication, Emergency Support, Smart Parking Solution, Vehicle Owner Alert, Urban Traffic Flow, Misuse Prevention, Public Assistance System.

I. INTRODUCTION

Vehicles can be seen in all the places and in these rapidly developing cities alongside streets of markets, residential areas, office districts, and public buildings. Due to which, problems have become so frequent that if a vehicle is left in an inappropriate place, it blocks the way, causes traffic jam, or even is abandoned in an emergency situation. The difficulty of contacting the owner arises because their phone number is not known. There are people who argue, wait, and even call the police, as there is no method to quickly and safely notify the owner.

To solve this issue, an Android application named "Public Vehicle Notification System" has been developed. Instead of a phone number, the system uses a digital vehicle identification tag or a QR code. Anyone who is in an unfortunate situation could be a traffic officer, a security guard, or a local resident; by just scanning the code or entering the ID in the app, they can send a message to the owner.

A secure cloud server is the medium through which all the notifications are routed, which ensures that the owner's private data will remain confidential. The app enables the users to select a message from the available options like “Your vehicle is blocking the pathway” or make a custom written alert as per the circumstance. There is also a provision of the system requesting a simple login or verification from the person sending the alert so as to prevent the misuse of the system.

In addition to dealing with everyday parking issues, the system is also available for use in emergencies and judicial work. To sum up, this app acts as an ingenious and secure way of communication between the public and car owners. It lowers the chances of conflicts, facilitates the flow of traffic, and is instrumental in building a well-organized urban environment.

II. LITERATURE SURVEY

Numerous academics have looked into how contemporary technologies, like intelligent vehicular networks, Android-

based systems, and QR codes, can improve parking control, vehicle monitoring, accident response, and communication in transportation settings. The main conclusions from the most notable studies in this field are summarized in the section that follows.

2.1. Advanced Vehicle Identification System Using Windshield-Integrated QR Code Technology Osama W. Ata and his team members [1] introduced an enhanced vehicle identification approach that uses QR codes mounted or pasted on the vehicle's windshield. In their design, the QR code functions as a digital identifier, allowing cameras or scanners to effortlessly and accurately recognize the vehicle. Their research focuses on safe and unchangeable identification, supporting automated verification tasks and law enforcement operations. This study is similar to ours in that it links a car to its digital profile using QR-based technology.

2.2. Live Emergency and Warning Alerts Through Android Application for Vehicular Ad Hoc Network Communication M. Milton Joe and B. Ramakrishnan [2] came up with a new way to communicate emergency and warning messages by using Android-based systems through Vehicular Ad Hoc Networks (VANETs). Their method facilitates fast communication between vehicles and roadside units, especially in situations which are at a turning point. Consequently, it is ensured that safety warnings get to the closest drivers without delay. In their research, they primarily concentrate on the communications of V2V and V2I through VANET technology; however, they emphasize that the most instant notifications on mobile devices can significantly contribute to the decrease of accidents on the road. Their concept is equivalent to the quick delivery of messages to vehicle owners in our system. **Reporting Module (reporting.js)**

2.3. QR Code based Smart Parking System Vedant Deepak Dokania and his team [3] developed a Smart Parking System which employs QR codes for a lot more convenient parking. Their method enables users to check in and check out by just scanning, identifying themselves, and even being automatically **WebSocket Module (websocket.js)** given a parking spot. It reduces the intervention of the staff to a minimum and it is very simple for people to find and reserve available places. Their study is an excellent demonstration of how QR codes could be a game-changer in the parking field. Besides that, the application of QR codes is very economical and userfriendly—basically, anyone can install it for different vehicle systems.

2.4. QR Code-based Real Time Vehicle Tracking in Indoor Parking Structures Kanumuru Rajesh and his team [4] have invented a real-time vehicle tracking system for indoor parking lots, which makes use of QR codes for location identification. They made the decision to install QR tags at the selected points around the parking area, so that once you park your car, you simply scan the code that is closest to you. Afterwards, you perform the scanning one more time to get the exact location of your car. It is an easy solution to that problem which is very common but you seldom realize—losing track of where you parked in the big indoor garages. Their study demonstrates how QR-based localization is a great relief for users in cramped areas and this is in agreement with our concept of employing QR codes for the speedy retrieval of vehicle information.

Interactive Android-Based Indoor Parking Lot Vehicle Locator Using QR-code The team led by Siti Fatimah Abdul Razak [5] developed an Android application that locates cars in indoor parking lots by the use of QR codes. The process is as follows: you park your vehicle, scan a QR tag that is close to you, and the application keeps the location of your car. When departing, the application guides you back. Their method, which is a mixture of QR codes and a straightforward mobile interface, is a great way to solve the problem of those perplexing indoor parking areas. This is another nice instance of how smartphones and QR codes can alleviate the pain of the daily routine.

2.5. Management of Road Accidents in Cities Through Smart Technologies Mithun R and Rupavahini S [6] investigation how the use of intelligent technology could facilitate the management of road accidents in urban areas. For this, they created a device equipped with sensors and a communication unit that is able to detect a collision and send an emergency call to the closest responders or the right authorities, in the shortest time possible. Such rapid, automated alerts are therefore instrumental in drastically reducing the time of intervention in accident situations.

III. GAPS IDENTIFIED

Researchers have gone through a wide range of topics—vehicle identification, smart parking, indoor vehicle tracking, accident detection, emergency alerts, and so on. To make the transportation system smarter and more user-friendly, they have combined various technologies like QR codes, Android apps, wireless networks, and VANETs. However, a few practical, daily problems still exist and have not been sufficiently addressed by the research conducted so far. One of the big reasons is that there is still no easy, safe, and anonymous method for people or authorities to contact vehicle owners in such cases when a car is blocking the way, parked in a wrong place, or involved in a minor incident without the need for the police.

1. Focus of Existing QR Code-Based Systems Osama W. Ata et al. have illustrated the incredible convenience of QR codes for vehicles and their users. Typically, these are the codes that you can find either attached to the glass of the vehicle or are integrated into the parking structures. They facilitate activities such as rapid identification checks by the police, straightforward access and departure in parking areas, user verification, and ensuring that parking spaces are utilized efficiently. Additionally, there are some installations that use QR labels to guide people to their vehicles in those perplexing indoor parking areas. All of this research leads to the conclusion that QR codes are one of the most reliable, inexpensive, and the easiest ways to implement solutions. But here is the thing—so far, the focus has been on paperwork and making parking easier. No one has really tackled the problem of creating a safe, anonymous way for regular folks to let a car owner know if their car's blocking someone or causing trouble. That need still has not been addressed.

2. Limitations of Existing Android and VANET-Based Alert Systems When people train big Vision-Language Models (VLMs), they usually grab massive datasets from the internet—stuff like raw, unfiltered image and alt-text pairs. It's fast, and it lets the models go through tons of data. But there's a trade-off. Study after study points out that all this noise—random errors, weird labels, plain old junk—makes the models less accurate and harder to figure out. The issue is, noisy data creates all kinds of messy connections. Take "vehicles," for example. The same word might show up next to thousands of completely different dishes.

That kind of confusion gets baked into the model, making it struggle to really understand the details it needs for sharp, reliable search results.

3. Privacy and Security Issues in Contacting Vehicle Owners Privacy and security are still a big problem with the way things work right now. You see it all the time: people leave their phone numbers on their windshields, stick notes on the dashboard, or just toss a visiting card inside the car. Sure, it makes it easy for someone to call if there is an issue, but this comes with a bunch of headaches. When you put your phone number out there for all to see, you are basically inviting spam calls, random harassment, or even worse someone misusing your details. Once that number's out in the open, you cannot control who saves it or what they do with it. Additionally, there is no real way to know who's trying to contact you. Anyone can pretend to be a parking official or some authority and try to scam you. Even if you want to stop being contacted, your number stays there, stuck on the glass, long after you have changed your mind. From what's been observed in the research done, no one's really perfected a way to keep owners' details private while still letting people reach them securely when it matters. The QR code systems we sometimes see are usually just link to basic information like a digital ID, the license plate, or a parking spot. They do not give you a safe, private way to communicate. Those accident detection systems are just send alerts to a few trusted contacts or to official servers they do not help if someone in the parking lot needs to get in touch about a blocked car or something similar. So, the gap's still there. Still there is no safe system ensuring data confidentiality.

4. Lack of Two-Way, Context-Aware Communication Many of the communication systems used in vehicles and the technologies supporting smart mobility are still largely one-way in nature. To illustrate these with examples, let's take accident detection technologies - when a crash occurs, the car automatically contacts emergency services, thus sending out a distress signal or location data - but the interaction practically terminates at that point. In the same way, VANET (Vehicular Ad-hoc Network) structures disseminate warnings or alerts to whoever is around, however, these messages are simply outgoing announcements that do not come with any built-in response or dialogue mechanisms. Moreover, characters such as intelligent parking that are dependent on the usage of QR codes or the like, simply send the information regarding a vehicle's arrival or departure to a central server; thus, a real exchange of information between users or between users and the system is not possible. The absence of two-way communication is a considerable disadvantage, especially when the need for real-time responsiveness and situational awareness is high. In the "real" world, authorities and automated systems cannot solely push out information; they have to receive confirmation that the recipient has, in fact, obtained, understood, and most probably executed the instructions. As an example, in case of an emergency, first responders may need to be sure that the drivers have already been informed about road closures or detours and are able to recognize these directions on the spot. Similarly, drivers would be better off if given such opportunities to interact with others having the ability to raise their concerns, offer updates, or get the required help, rather than being mere passive recipients of information. To close this divide necessitates a paradigm shift of how vehicle communication systems are designed. There is an unambiguous necessity for platforms that allow real-time, bilateral communication so as to make possible direct and meaningful interactions vehicle owners, officials, and other stakeholders. However, the implementation of such systems is accompanied by particular challenges. These solutions have to protect the privacy of users and guarantee the security of data since personal and location information is inevitable. The interfaces required should be user-friendly and less visible, thus, allowing users to communicate with ease even under difficult situations or while driving. Besides that, they ought to be strong, capable of handling the growth of users, and compatible with the existing infrastructure without resulting in excessive complexity. In the end, giving vehicles and their operators the ability to actually have a conversation with them which is far better than just the passive sharing of information will lead to the development of more intelligent, safer, and quicker hydrous transportation networks.

5. Integration of QR Codes with Cloud-Based Messaging for Public Use A major drawback of the current technologies is that they cannot easily link QR codes with cloud messaging services, such as push notifications, in a user-friendly manner for people in general. It is also worth mentioning that some Android applications have the feature of sending alerts, but these are normally used in certain areas like private parking lots or vehicles with a network connection, and hence, situations involving the general public are out of their reach. As a rule, all these mechanisms are designed for certain specific localities and their scope is not large enough to cover, for example, a whole city. Take for instance the issue with a person who is walking past parked cars and wishes to notify the owner about some problem- in the ideal case, he/she just needs to scan QR code, which sends a message to a secure cloud server, which then routes that message in order to notify a particular car owner's mobile device while taking care of both parties' privacy and data protection. At this point in time, vehicle-related QR codes mostly concern controlled access areas, such as checkpoints for police or administration of private properties. Public-facing, reliable, and simple-to-use solutions that facilitate communication between strangers without giving away their sensitive information are almost nonexistent. The absence of a uniform, open system at the core of this issue means that ways of urban mobility to which people can comfortably and effortlessly communicate are seriously hindered. There is also no simple car driver alert method which is not a trespassing or inconvenience, so that people may warn a car owner of issues like headlight on or blocking a driveway in an unintrusive manner. The novel system is intended to solve these problems by incorporating familiar elements such as QR codes, apps, and notifications via a cloud server but rather cleverly thinking of their usage in daily-life interaction scenarios. Establishing an environment where one can be anywhere in a metropolis and access a car owner's contact details by scanning just one QR code on him/her is an enabling tool for solving common parking or car-related incidents in the right way, whereby respect of involved persons and privacy is retained simultaneously. Besides, the solution features strong points concerning safety and privacy measures, which guarantee that neither one will be able to access the other's confidential information during the procedure. This technology-driven platform has resources to liberate metropolitan living by, for instance, making car-related communication doable anytime and anywhere in the city, which is a big leap on the part of technological convergence that is still stubbornly divided in urban settings.

IV. SYSTEM ANALYSIS

This chapter is about the deficiencies of visual search technologies that make up the Existing System, which is the collective term for these technologies. While such systems are efficient in some environments, they have issues through accuracy, scalability, privacy, and interactive communication, among others. Through the detailed review of these weaknesses, a bigger picture of problems these technologies have is

drawn up. After the discussion of these issues, the chapter describes the design and the goals of our Proposed System as a response to the limitations of the Existing System. The Proposed System is equipped with such new features as QR code usage, cloud-based messaging, and secure, privacy-friendly communication channels, to name a few that are aimed at improving usability, real-time interaction, and public accessibility. By comparing the Existing System with our solution, this chapter creates the opportunity to understand that the proposed way is more convenient, socially beneficial, and technically resilient .

4.1. The Existing System Most of the time, parking violations and vehicle notification systems that are not digitally connected are done manually, and the activities take a reactive approach. In short, when a car is parked in an illegal parking space, blocking a gate, or causing an obstruction, officers usually put a paper on the windshield. The vehicle owner is informed of such a situation whenever he comes back to the vehicle and sees the ticket. What if the owner does not check the vehicle for several hours or even days? The obstruction is there, and there is no real-time communication channel between the reporter and the driver. As a result, other users of the affected space get delayed solutions and experience frustration. There are organizations that have made attempts to improve the process by using centralized SMS alerts. These systems maintain a database of vehicle numbers and the phone numbers of the owners, and when an issue is raised, bulk SMS messages are sent to a number of registered drivers. This method is inaccurate and expensive. Many recipients get messages that are not relevant to them, while the actual violating driver may still overlook or ignore the alert. Since each SMS has some monetary cost, if mass messaging is done frequently it will increase the organization's expenses very quickly and thus will not be sustainable for frequent or large-scale usage. On the backend, existing systems often rely on static databases and unstructured file storage for reports and images. Photographs of violations, descriptive notes, and associated metadata are stored indefinitely without a clear retention policy. Over time, this leads to storage bloat: databases grow excessively large, file systems accumulate outdated evidence, and backups and queries become slower and harder to manage. Manual cleanup of old data is error-prone, forcing administrators to either avoid the task or risk deleting valuable information.

These characteristics result in several key limitations:

• **Delayed Notification** – Physical slips are discovered late, and SMS alerts may not be read immediately, causing a long delay between the occurrence of the violation and the owner becoming aware of it.

• **High Operational Cost** – Dependence on SMS as the primary notification channel generates recurring expenses. Bulk or broadcast messages further increase costs without ensuring that the correct driver receives a targeted alert.

• **Storage Bloat** – Reports and images accumulate without automated cleanup, storing data beyond its useful lifetime, wasting disk space, and gradually reducing system performance.

• **Poor User Experience** – Reporters receive no real-time confirmation or feedback, while vehicle owners get either delayed physical notices or generic SMS messages with minimal context. There is no smooth, two-way interaction between the reporting party and the vehicle owner.

These limitations emphasize the need for a more intelligent, real-time, and resource efficient notification system capable of delivering targeted messages, managing data effectively, and providing a better experience for both reporters and vehicle owners

4.2. The Proposed System We have engineered a modern, full-stack notification solution to supersede conventional, unreliable, and latency-prone vehicle alert mechanisms. Our system delivers a robust, efficient, and easily maintainable platform leveraging real-time communication, resilient data handling, and containerized deployment.

I. Dual-Channel Delivery for Maximized Reliability The core of our platform is a two-tiered notification strategy that ensures message delivery while optimizing for speed and cost. • Primary Channel: Real-Time WebSocket Alerts The backend prioritizes immediate alert dissemination via a persistent WebSocket connection when an active driver session is detected. This mechanism guarantees zero latency notifications to users engaged with the application, maximizing responsiveness and enhancing the user experience. • Secondary Channel: Guaranteed SMS Fallback If the primary connection is inactive (e.g., user is offline, unregistered, or the connection is lost), the system seamlessly transitions to a third-party SMS service (e.g., Twilio). This systematic session check ensures the cost-effective, real-time method is always attempted first, maintaining high delivery reliability while controlling operational expenditure.

II. Advanced Connection Resilience and Persistence To mitigate the impact of transient network failures, we implement a dedicated message persistence layer. In-Memory Message Queues An in-memory, per-user queue (mapping $userId$ to $PendingMessages$) is maintained to handle connection interruptions. Any notification intended for a user experiencing a dropped connection is immediately stashed in their respective queue. Upon connection reestablishment, the system automatically triggers a batch delivery of all queued messages, guaranteeing zero message loss due to short-term network instability.

III. Structured Evidence and Data Lifecycle Management The platform incorporates secure media handling and a disciplined data retention policy.

Secure, Streaming Image Ingestion The backend utilizes streaming upload techniques (e.g., Node.js/Multer integration) to safely process evidence images, supporting files up to a defined limit (e.g., 10 MB). Each image is cataloged with a unique, auditable identifier following the format [VehicleID] [Timestamp].jpg (e.g., ABC123_1717698042.jpg), facilitating efficient linking to reports and subsequent automated cleanup.

Automated Data Retention and Cleanup A scheduled, nightly job is executed to enforce data governance standards. This task performs a systematic sweep of both the persistent storage (database) and the file system (uploads directory), purging all reports and associated media older than the defined retention window (e.g., seven days). This process ensures storage optimization and continuous compliance with data privacy requirements.

IV. Containerized Deployment and Operational Simplicity The entire solution is architected for simplified deployment, maintenance, and scalability.

Microservice Architecture The system is deployed using containerization, separating the Node.js application layer and the PostgreSQL persistence layer. Services communicate via a dedicated private network.

Zero-Downtime Initialization Startup procedures leverage health checks to ensure application containers only initialize after

the database schema has been successfully provisioned via an initial script (e.g., init.sql). The entire stack is managed and launched via a single orchestration file (e.g., Docker Compose), ensuring consistent, repeatable, and painless deployment across all environments.

4.3. Objectives The design and development of this system are guided by five core objectives, each directly addressing critical limitations observed in traditional vehicle notification and parking violation tools. These goals serve as the architectural drivers that shape the system's overall structure, reliability, and security.

4.3.1. Hybrid Notification and Resource Optimization The primary objective is to implement a tiered notification delivery mechanism. The system will prioritize real-time alerts utilizing Web Sockets for immediate user communication. In the event of a client connectivity failure or an unsuccessful real-time transmission, the system will execute a graceful failover to a more resource-intensive channel, specifically Short Message Service (SMS). This approach ensures guaranteed message delivery while simultaneously minimizing operational costs associated with SMS usage.

4.3.2. Enhanced Reliability via Resilient Connection Management We aim to mitigate the impacts of common network instability issues, such as dropped connections and message duplication. This will be achieved through the integration of inmemory queuing mechanisms. The queues will buffer messages, ensuring that all notifications are persistently handled and ultimately delivered, maintaining data integrity and guaranteeing message throughput even under conditions of network degradation.

4.3.3. Data Governance and Controlled Retention Policy A critical objective is to enforce stringent data security and privacy protocols through controlled data lifecycle management. A mandatory time-based retention limit (e.g., seven days) will be established for all sensitive assets, including violation reports and associated images. This policy is designed to prevent the unauthorized accumulation of sensitive data and to ensure compliance with privacy best practices.

4.4.4. Automated Data Lifecycle Management To support the retention policy, the system will incorporate a fully automated lifecycle management process. This eliminates the need for manual intervention by integrating mechanisms that automatically purge aged records and image files. This automation is essential for maintaining storage efficiency, optimizing system performance, and ensuring operational scalability as the user base expands.

4.4.5. Simplified Deployment and System Portability The final goal is to enhance the system's deployability and portability. This will be accomplished by containerizing the entire application stack, including the backend service and its associated database. Utilizing a single-command orchestration tool (e.g., Docker Compose) will allow users to initiate the entire environment quickly, significantly simplifying setup, replication, and maintenance across various operational settings.

V. SYSTEM REQUIREMENT SPECIFICATION

This chapter outlines the technical hardware and software requirements needed for the development, training, deployment, and day-to-day operation of the proposed visual search system. The requirements are categorized into two major sections: hardware specifications, which cover both server-side and user-side devices, and software specifications, which include the tools, frameworks, and execution environment required for building and running the system.

5.1. Hardware Requirement Specification

5.1.1 Server Requirements • CPU: A multi-core processor (minimum 4 cores recommended) • RAM: At least 8 GB, with 16 GB preferred for production environments • Storage: Minimum 50 GB, preferably on an SSD to improve database performance • Network: A reliable internet connection to maintain WebSocket sessions and interact with external services such as the Twilio API
.

5.1.2 User Device Requirements (Client) • Mobile Application: Compatible with Android devices running version 8.0 or higher, or iOS devices running version 12 or above • RAM: Minimum of 2 GB • Network: Connectivity via 3G, 4G, 5G, or Wi-Fi for smooth app operation.

5.2. Software Requirements.

5.2.1 Operating System Server: Linux (Ubuntu 20.04 or later), Windows with WSL2, or macOS Client: Android or iOS operating systems 5.2.2 Container Runtime • Podman: Version 3.0 or higher • Docker: Version 20.10 or higher • Compose Tools: podman-compose or docker-compose v2.

5.2.3 Programming Language • Backend: Node.js (JavaScript), Express.js framework, WebSocket communication • Frontend: XML for Android UI layouts, JSON for data exchange, Java for Android application logic • Database: PostgreSQL.

5.2.4 Development Tools Used • Android Studio: Main IDE for developing the Android application, UI layouts, and related components • PostgreSQL: Relational database system to store driver information, vehicle details, and report records • Node.js: JavaScript runtime used for implementing the backend server • NPM (Node Package Manager): For installing backend libraries, modules, and dependencies • Git & GitHub: Version control tools for managing code revisions and maintaining a remote repository for collaboration and backup.

5.2.5 Additional Software • Twilio API Dashboard: Used for SMS integration, monitoring message logs, and managing API configurations • Docker & Docker Compose: Containerization tools for running Node.js and PostgreSQL services in isolated, portable environments.

VI. SYSTEM DESIGN

The base design of the system to be built can be imagined as a multistage, modular pipeline that enables a smooth flow of activities starting from the initial data ingestion to real-time output generated by the visual search interface. The following chapter details the technical aspects of each separate module, describing the data flow throughout the system in great detail. Such a hierarchical approach is vital for providing the efficient flow of data and the correctness of the final result. Moreover, the system's modular architecture directly improves qualities of the system such as scalability, serviceability, and the possibility of adding

new features without changing the core architectural framework.

6.1. Modules

The system is organized into five major modules, each responsible for a specific aspect of the application's functionality. Together, these components enable smooth user registration, violation reporting, real-time communication, secure data storage, and centralized server coordination. Each module is designed to operate independently yet integrate seamlessly with the others, ensuring a stable and predictable system workflow.

6.1.1. Registration Module (registration.js) The Onboarding of New Vehicle Owner is handled by the Registration Module. It checks that the provided user information is valid, creates a new user id and stores the registration details safely in Database. The module also connects each user's details such as vehicle number, email ID, phone number etc. to a digital identity or a QR code. It also verifies existing records to avoid redundant accounts. Upon successful registration, a confirmation message is returned to the master server that only verified users Knowing Receiver (KNR) will be added.

6.1.2. Reporting Module (reporting.js) The Reporting Module receives and reviews violation reports from citizens or authorized parties. It receives basic information like the offence message, location, time-stamp and optionally attached images (e.g. pictures of a parking violation). When the notice is reported to the module, it queries database and finds out who was in charge of that vehicle at that time, then immediately sends a notification by WebSocket with SMS fallback.

6.1.3. WebSocket Module (websocket.js) This is the module that takes care of the entire real-time, two-way communication from the server to your users and back. After establishing a connection with a client, the module initiates a persistent WebSocket where user IDs are utilized in such a way that each session is given its unique user ID. In order to offer dependable message delivery, it keeps records of active connections, carries out periodic heartbeats and holds messages if the recipient user is offline. Notifications that have been queued are sent to the user as soon as he or she reconnects. This module is what makes it possible to have real-time notifications and is instrumental in increasing the system's overall responsiveness.

6.1.4. Database Module (db.js) The Database Module is responsible for the connection pool to the PostgreSQL database and results in all queries being efficient. It reduces the response time and prevents the server from being overwhelmed at high traffic, by the use of re-pooled connections. It offers the features for the core database operations like user registration, vehicle finding, report storing, and notification logging. The module safeguards data from corruption by input validation and ensures that all system data are even in the long run stored in a reliable storage facility.

A

I

6.1.5. Main Server Module (main.js) The Central Server acts as the heart of the system. It creates all API endpoints -- for signing up, logging in & real-time messaging -- and configures items like CORS preflight requests and authentication hooks. The server compositely controls all other modules and makes sure that date travels smoothly between registration, reporting, communication and storage. It also does global error handling and managing overall application lifecycle.

6.2 Database Design

6.2.1. Driver

The Driver entity holds the personal and identification information of a registered owner or driver of a vehicle. It creates the base of user information layer in system, guarantees each driver to be unique identified on application. Attributes: • Driver_id – Unique id for the driver in system.

• Driver_name: Full, legal name of the driver. • Vehicle_number – The vehicle number registered with the driver is the foreign key that refers to the Vehicle entity. This entity guarantees that each driver is recorded as a unique entity, securely and uniquely linked with its vehicle, and traceable whenever registration/reporting/notification operations are triggered in the system.

6.2.2. Vehicle

The Vehicle entity represents all car vehicles in the system, and this entity is the center pivot between reported or notified reports, i.e. from "Reports" you can flag a vehicle for notification when appears in an orange few cars to avoid them. Attributes: • Vehicle_number – The unique registration number assigned to the vehicle (Primary Key) and its main identity. • Driver_ph_no - The driver's or registered owner's mobile no associated with the car. This entity stores essential vehicle-related details, ensuring that every vehicle is uniquely recognized and its owner can be reached immediately for alerts, updates, or violation related notifications.

6.2.3. Reports

Reports All violations or complaints entries entered by the citizenry or authorities are stored to the Reports entity. It serves as the central repository of all complaints and retains text as well as photo documentation related to each incidence. Attributes: • Report_id – A unique id associated with each report. • Vehicle_number – The vehicle for which the report is; (Foreign Key referencing to Vehicle). • Report_msg – The text message information or descriptive name explaining what the violation is. • Image_filename – Uploaded photo evidence Filename if saved. • Created_at – The exact time that the report was created. This gives the opportunity to attach several reports to one and the same vehicle making possible a complete temporal order report history for specific tracking and analysis.

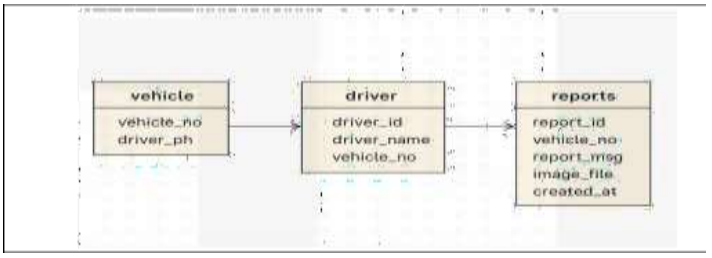


Fig 6.2.1 Database Schema Design

VII. IMPLEMENTATION

This chapter features the entire creation process of the Public Vehicle Notification System, the different technologies used, the backend logic coded, the database structure designed, and the mobile application developed. The description is about how the

blueprint for the system was converted into a functioning secure and efficient solution. The goal of this implementation round was to develop a working system that could provide real-time notifications, process image-based violation reports, and facilitate anonymous communication between the public and vehicle owners. Moreover, the chapter points out the measures taken for steady system operation, dependable data handling, and easy release through containerized services. **Technologies Used**

Well-structured technologies stack divided into backend, frontend, DevOps, and services integrations are used in building this system.

7.1 Frontend (Mobile Application) A mobile app acts as the primary interaction for the users to scan a QR code and report. It is built using:

a. Android (Java/Kotlin) The business logic within the app is written in Java or Kotlin. These languages allow the app to effectively handle crucial functions such as: • Confirmation of report content (with the message, vehicle ID, etc.) • Taking the photo with your device camera • Inputting data through the UI and sending it as REST calls to the backend • Keep the WebSocket open for Push notifications • Processing and reading QR code to facilitate faster reporting Both Java and Kotlin integrate seamlessly with Android libraries, making sure that the application remains responsive, lightweight, and easy to maintain.

b. XML Layouts XML was used to construct all the user interface components of the application : • The reporting screen • The image upload interface

b. Express.js Express.js is the central backend framework by which all API routes are defined and managed in the system. It is the main tool that organizes server logic, request handling, and middleware application in a neat, clean structure. Essentially, the web app interacts with the key routes to perform the main functions it is designed for:

- **Register** – Driver Registration This API point is about getting the info on a driver, checking it, and then saving the data safely in the database to create a new user profile.
- **Report** – Report Submission With this path, users are enabled to file a complaint about a violation, giving the vehicle number, message, and optionally, an image as the proof.
- **Get-reports** – Report History Retrieval This location in the API will get all of the reports in the past that are associated with either a vehicle or a driver so that the user can look at the old submissions.
- **View-image** – Display Evidence Image This URL is used to get the image file that is linked with a report so that the user can see the uploaded evidence. Express.js also makes it easier to add middleware and allows a modular code structure, which means that the backend can be more efficient, maintainable, and scalable.

c. Multer Multer is used to manage the image uploads while making sure it's secure on the server. It provides: • File validation, approval only for the available image formats. • Secure file storage, placing all uploaded images in /uploads folder. • Auto file naming, files are saved using vehicle number and time stamp combined. • These features ensure that the images in such a system remain distinct, well organized and easily traced.

d. PostgreSQL + pg Library The system's primary relational database is PostgreSQL. It is retrieved using the pg library in Node. js, which supports: • Parameterized Queries that allow for safe transactions with the database. • Optimized insert, update, and fetch handling of users, vehicles and reports. • You need to sanitize your incoming data for SQL Injection. • Connection pooling: be more performance efficient by reusing opened database connections.

e. WebSocket (express-ws) WebSocket integration opens several key features in the system: • Push notifications in real time to receive alerts immediately upon release. • Continuous communication between server and drivers are of permanent two-way, keeping the app connected even if it's online. • Stop continuous polling which is possibly reducing unnecessary server communication and improving performance. • Very fast message delivery and very low bandwidth usage, making communication so easy and light. These features ensure that any connected drivers are directly notified of violations.

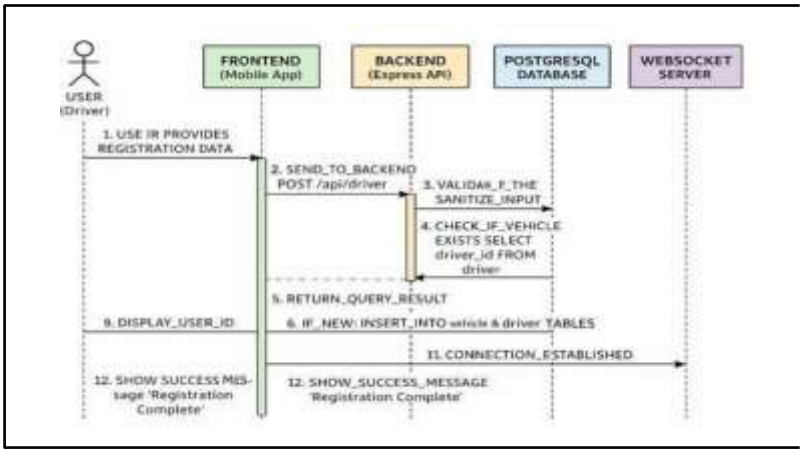


Fig 7.1 Driver Registration Sequence Diagram

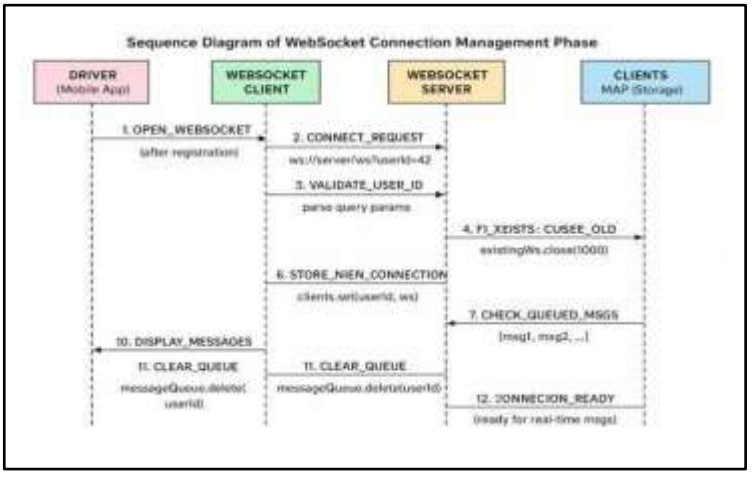
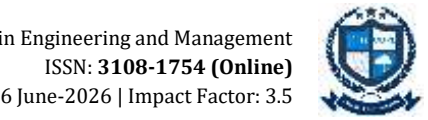


Fig 7.2 WebSocket connection management sequence diagram

VIII. SYSTEM TESTING

Types of Testing Performed

8.1 Functional Testing Functional tests have been implemented to check the operations of each system features in accordance with the requirements. • Driver Registration API The register path was



experimented with various input combinations—e.g., different driver names, contact numbers, and vehicle IDs—to verify that validation rules were enforced properly and that the data were inserted accurately into the database. • Report Submission with Image Upload The report destination was tested to confirm that not only text-based reports but also image uploads are handled properly, saved on the server, and referred to the correct vehicle entry. • WebSocket Real-Time Communication The WebSocket mechanism was checked by carrying out live communication of messages. Messages were sent to make sure that notifications show up immediately when the driver is online and that they are saved in a queue when the user is offline. • SMS Fallback Using Twilio API The testing of the fallback method was done to be sure that in case the driver is not connected via WebSocket, Twilio will send an SMS notification to the mobile number which is already registered without any manual intervention.

Fig 8.1 Driver Registration

8.2 Integration Testing Integration tests were performed to check the functionality of all the interconnected parts of the system without resulting in an error or data inconsistency.

Android App ↔ Backend Server Checked the interaction of the mobile app with backend APIs, which included confirming JSON responses were correct, errors were handled properly, data transfer was secure, and user requests were processed accurately.

Backend ↔ PostgreSQL Database Made sure driver registration, report creation, and report history retrieval were operations going through smoothly and that the database was returning consistent results without any query failures.

Backend ↔ Twilio API Ensured that SMS alerts are only sent when there is no WebSocket delivery confirming that the fallback method is functioning exactly as it is supposed to.

Backend ↔ WebSocket Clients Checked that a WebSocket connection is established, reconnections are handled, and queued messages are delivered to a user when he/she is online again.

Image Upload Module ↔ File System Confirmed that images uploaded are stored correctly on the server, can be retrieved without being damaged, and are shown properly via the /view-image endpoint.

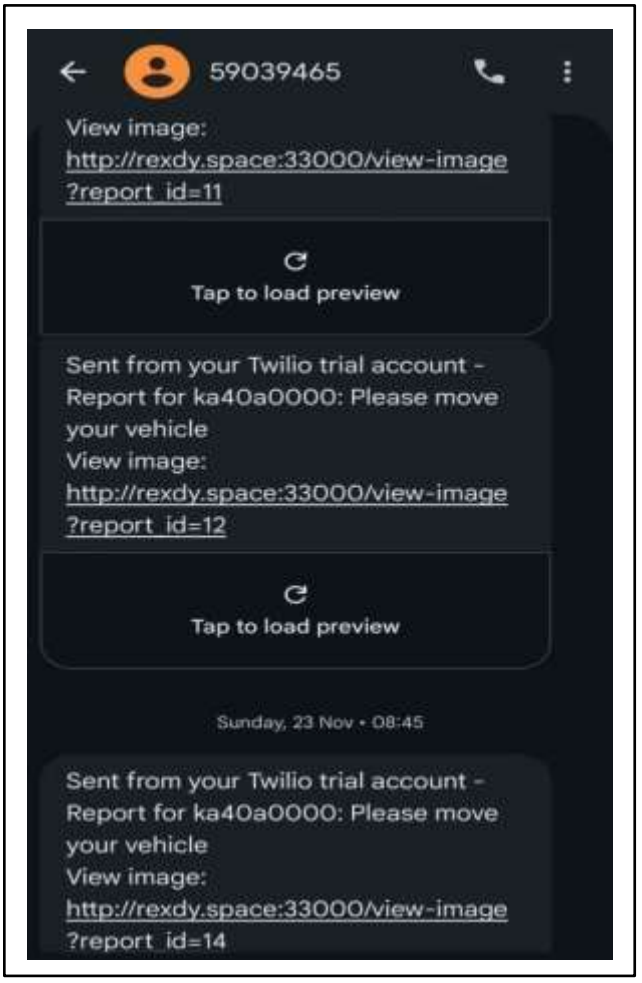


Fig 8.2 SMS through Twilio API

3.Security Testing To make sure the system is safe, it was checked how it handles incorrect and even malicious inputs from users.

SQL Injection Prevention To prevent attackers from inserting harmful SQL code, all database operations are done with parameterized queries, which is the method that is used here.

Invalid Image Format Filtering The backend must be sure that the files are of the correct MIME type and if it finds a file that is not safe or is not supported it (for example, a .exe file or a damaged image) will refuse it.

Phone Number Verification The registration component that is responsible for validation of the 10-digit mobile number ensures that there are only correctly formatted numbers.

Rate Limiting & Anti-Abuse Controls The API endpoints were hit with the repeated and rapid requests in order to find out if the server is being stressed. Docker logs show that the excessive or spam requests are being dropped automatically**.**

Performance Testing Performance testing was conducted to assess how the system behaves under high load, including multiple simultaneous users, repeated notifications, and large data

transfers. This evaluation ensures that the application remains stable, responsive, and efficient even during peak usage scenarios. **Concurrent WebSocket Handling** The test was performed with more than 100 users at the same time to check whether the communication in real-time can be kept stable and scalable**.**

SMS Fallback Stress Testing Repeated offline scenarios were simulated to see how the system would handle multiple Twilio SMS retries if there were no delays or failures.

Large Image Upload Evaluation It was confirmed that image files of up to 10 MB are uploaded in a timely manner, compressed properly, and stored without any performance impact.

Database Query Load The system response was evaluated when several users at the same time carried out registration, report submission, and history retrieval in order to assure low latency.

API Response Under Heavy Traffic All REST endpoints were tested under many requests within a short period of time in order to verify that the performance was consistent and that there were no timeouts.

Server Resource Monitoring CPU, memory, and storage usage were monitored under the heaviest load to confirm that the backend was stable and that it did not crash.

Fig 8.3 Reporting to the owner

IX. REFERENCES

[1] Advanced Vehicle Identification System Using Windshield-Integrated QR Code
Technology: Osama W. Ata, Abdallah M. Abu Munshar, Ahmad S. Saa'd, published on
IEEE between 2006 and 2018.

[2] Live Emergency and Warning Alerts Through Android Application for Vehicular Ad
Hoc Network Communication (Android VANET), Wireless Personal Communications:
M. Milton Joe, B. Ramakrishnan (2021).

[3] QR Code based Smart Parking System", International Conference on
Communication and Signal Processing: Vedant Deepak Dokania, Mayur M. Sevak, Dev
Dineshkumar Patel and Pratham Shriharsh Barve (2020).

[4] QR Code-based Real Time Vehicle Tracking in Indoor Parking Structures Proceedings
of the Second International Conference on Intelligent Computing
and Control Systems : Kanumuru Rajesh, K.V. Mounika Reddy, S.Swaranalatha, M.Supraja (2018).

[5] Interactive Android-Based Indoor Parking Lot Vehicle Locator Using QR-code:
Siti Fatimah Abdul Razak, Choon Lin Liew, Chin Poo Lee, Kian Ming Lim ,2015
IEEE Student Conference on Research and Development (SCOReD).

[6] Management of Road Accidents in Cities Through Smart Technologies, Accident
Recognition and Alarm System s and respond to the notification
which would be sent through our accident.

[7] Car accident detection and reporting system: Mandeep Kaur Angrula, Mrs. Narinder
Kaur, 06 Issue: 12 Dec 2019.

[8] IOT Based Automatic Vehicle Accident Alert System: Nazia Parveen, Ashif Ali,
Aleem Ali , published on IEEE in 2023.

[9] Using Smart phones to Detect Car Accidents and Provide Situational Awareness to
Emergency Responders". Chris Thompson, Jules White, Brian Dougherty,
Adam Albright, and Douglas C.Schmidt, Vanderbilt University, Nashville, TN
USA, published on IEEE in 2012.

[10] Smart Automatic Vehicle Accident Detection, Tracking and Messaging System using
GPS and GSM: Keshwarya kunjekar, Prashant Karad, Prof. V.S. Gawali in IRJET on
Feb 2019.

[11] Emergency vehicle notification system: Trevor Groves, Shaughnessy, published on
IEEE.

[12] P. Balios et al., "Motion Shield: An Automatic Notifications System for Passenger Safety," Sensors, 2022.

[13] J. Branquinho et al., "An Efficient and Secure Alert System for VANETs to Improve Road Safety," Sensors, 2020

[14] "A Robust Accident Detection and Notification" (IJARST/IJASTR style paper) — Android app integrated accident detection and multimodal alerting.

[15] "Smart public transportation system using Android app" (ResearchGate preprint) — integrated Android-based transit notification system.

[16] "Android-based real-time emergency alerts app (journal/impact report)" — overview/prototype of Android emergency notification systems for vehicles.

[17] "Public transport tracking system using IoT" (IJNRD) — GPS + Android real-time public transport monitoring.

[18] F. Cunha et al., "Data communication in VANETs: Protocols, applications and challenges," Computer Communications, 2016.

[19] "Destination alarm notification for public transportation passenger using Geo-Fence in mobile App," (geo-fence destination alarm mobile app).

[20] S. S. Dukare, "Vehicle Tracking, Monitoring and Alerting System: A Review" — review article covering tracking/alert systems and Android apps.