International Journal of Scientific Research and Engineering Development-– Volume 9 Issue 2, Mar-Apr 2026
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Electric Vehicle Recharge to Find Nearest Charging Station Using Web Application

Premkumar P*, K. Thenmozhi**
*(Department Of Information Technology, Dr. N.G.P Arts and Science College, Coimbatore, Tamil Nadu, India
Email: premkumar96682@gmail.com)
** (Department Of Information Technology, Dr. N.G.P Arts and Science College, Coimbatore, Tamil Nadu, India
Email: thenmozhi@drngpasc.ac.in)

----------**********----------

Abstract:

The proliferation of Electric Vehicles (EVs) creates an urgent demand for accessible charging infrastructure. However, current navigation solutions predominantly rely on paid proprietary ecosystems like the Google Maps Platform, which imposes high operational costs and data opacity on developers. This research presents the design and implementation of a Web-Based EV Charging Station Finder that operates entirely independently of commercial APIs. The proposed system utilizes a specific open-source stack: Leaflet.js for interactive map rendering, OpenStreetMap (OSM) for tile data, Nominatim for geocoding services, and the Leaflet Routing Machine (powered by OSRM) for turn-by-turn navigation. The backend is built using Node.js and Express.js with MongoDB as the database, while the frontend is developed using plain HTML, CSS, and Vanilla JavaScript — making the system lightweight and universally deployable without framework overhead. This paper evaluates the system's architecture, demonstrating how this open-source stack achieves low-latency proximity queries and zero operational cost, offering a scalable blueprint for sustainable EV infrastructure management.

Keywords: ***Electric Vehicle (EV), OpenStreetMap, OSRM, Leaflet.js, Nominatim, MongoDB, Node.js, Haversine Algorithm, Geolocation API.***

----------**********----------

I. INTRODUCTION

Electric vehicles are becoming more popular, but one major problem remains: drivers worry about running out of battery before finding a charging station. This worry, often called "range anxiety," stops many people from buying EVs. To solve this, drivers need simple tools that help them quickly find nearby charging stations and plan routes with charging stops along the way.

Most charging station finder apps today use Google Maps APIs. Google Maps works well, but it has two big problems. First, it costs money — every time someone searches for a location or calculates a route, the app owner has to pay Google. For small companies or students building apps, these costs add up quickly. Second, once you build your app using Google Maps, you become dependent on Google. If they change their prices or rules, you have no choice but to accept it.

This project solves both problems by building a completely free alternative using open-source tools. Instead of Google Maps, we use OpenStreetMap — a free map built by volunteers around the world. Instead of Google's routing service, we use OSRM (Open Source Routing Machine) which calculates routes just as fast, sometimes even faster. For displaying maps, we use Leaflet.js, a lightweight JavaScript library that works in any web browser.

Our system is deliberately simple. The frontend uses plain HTML, CSS, and JavaScript — no complicated frameworks like React or Angular. This makes the app load faster and easier to understand for any web developer. The backend uses Node.js

 ©IJSRED: All Rights are Reserved

with Express to handle requests and MongoDB to store station data. Anyone with basic web development knowledge can deploy and modify this system. This project proves that you don't need expensive commercial APIs to build a professional-quality navigation app for EV charging stations.

II. LITERATURE REVIEW

When we started this project, we looked at what other people have built and what problems they faced. We found that most EV charging station finders fall into two groups: those built with paid services like Google Maps, and those trying to use free alternatives.

A. Problems with Paid Mapping Services

Many developers choose Google Maps because it's easy to set up and works reliably everywhere. However, research by Phuangsuwan and others in 2024 showed that Google's pricing can be a serious problem [7]. They found that small organizations, universities, and non-profits struggle to afford Google Maps once their app becomes popular. Each map load, each address search, and each route calculation costs money. If 10,000 people use your app in a month, you might pay hundreds of dollars just for the map service.

Industry reports from early 2025 showed that Google increased their API prices again, making it even more expensive. This pushed many developers to look for alternatives. The problem isn't just cost — once you build your entire app around Google Maps, it's very hard to switch to something else later. You become locked into their system.

B. Problems with Free Mapping Tools

Free alternatives like OpenStreetMap exist, but they have their own challenges [2]. A study by Patel and Sharma in 2025 found that while OpenStreetMap is powerful, it's harder to learn than Google Maps [8]. The documentation is often confusing, and there are many different tools that do similar things, making it difficult to know which one to use.

Another study by Li and Wang in 2022 found that early websites using OpenStreetMap were often slow [9]. The maps took a long time to load, and moving around the map felt sluggish compared to Google Maps. This gave users a bad experience, even though the system was free.

C. What Was Missing

After reviewing existing solutions, we noticed a pattern: you could either have an easy-to-use app that costs money (Google Maps), or a free app that's hard to build and slow to use (early OpenStreetMap implementations). No one had successfully combined the best of both worlds.

Our project tries to fix this gap. We use free OpenStreetMap tools [2], but we build the system in a smart way that makes it fast and easy to use. We use modern JavaScript techniques and efficient backend code to match the speed of paid services without any subscription fees. We also keep the code simple — using plain JavaScript instead of complex frameworks — so that any web developer can understand and improve it.

III. PROPOSED SYSTEM ARCHITECTURE

The system is built upon a modular architecture where the geospatial logic is decoupled from the application logic.

A. Visualization Layer: Leaflet.js & OpenStreetMap

Unlike heavy mapping SDKs, the application uses Leaflet.js, a lightweight JavaScript library [3]. Leaflet does not contain map data itself; instead, it acts as a rendering frame. The map tiles are fetched dynamically from OpenStreetMap servers [2] using the standard tile URL pattern (https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png) . This separation ensures the application remains lightweight, loading tiles only as the user pans or zooms. The frontend is implemented entirely in Vanilla JavaScript and HTML [11], eliminating the need for a frontend framework and reducing bundle size and complexity.

B. Address Resolution: Nominatim

 ©IJSRED: All Rights are Reserved

To convert human-readable place names (e.g., "Chennai", "Delhi Airport") into machine-readable coordinates (Latitude/Longitude), the system interfaces with Nominatim, the search engine for OSM [2]. When a user searches for a location manually via the dashboard search box, Nominatim's API is queried using the fetch() API to pinpoint the exact coordinates on the map. The user can also use the browser's built-in HTML5 Geolocation API [10] (navigator.geolocation.getCurrentPosition()) to automatically detect their real-time location. Additionally, reverse geocoding is performed via Nominatim to display the human-readable address of the detected location on the map popup.

Note: When an administrator adds a new charging station via the Admin Panel, the latitude and longitude are entered manually into the form fields. The backend stores this coordinate data directly in MongoDB [5].

C. Navigation Engine: OSRM & Leaflet Routing Machine

The core routing capability is handled by two components:

- **OSRM (Open Source Routing Machine):** Used as the route calculation engine [1]. It utilizes Contraction Hierarchies, an algorithm that pre-computes travel paths, allowing it to calculate the shortest path across a road network in milliseconds.
- **Leaflet Routing Machine:** This plugin [3] acts as the bridge between the frontend and OSRM. It sends origin and destination coordinates to the OSRM server and renders the returned polyline (the route path) and turn-by-turn instructions directly onto the Leaflet map layer.

D. Authentication & Authorization

The system implements JWT (JSON Web Token) based authentication. Upon login or registration, the server generates a signed JWT token using the jsonwebtoken library. This token is stored in the browser's localStorage and sent as a Bearer token in the Authorization header for all protected API requests. Role-based access control (RBAC) is enforced via middleware — only users with the admin role can create, update, or delete charging stations.

E. Backend Architecture: Node.js + Express.js + MongoDB

The backend is built with:

- **Node.js** as the runtime environment [11]
- **Express.js** as the web framework for defining RESTful API routes [11]
- **MongoDB** with Mongoose ODM for data persistence [5]
- **bcryptjs** for password hashing
- **jsonwebtoken** for token generation and verification
- **dotenv** for environment variable management

Two primary data models are defined:

- **User Model:** Stores name, email, hashed password, and role (user/admin)

Station Model: Stores name, address, location (lat/lng), charging type (fast/normal/both), total slots, available slots, price per kWh, operating hours, amenities, and open/closed status

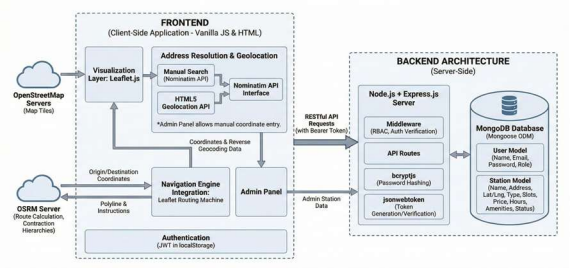

Fig. 1 Proposed System Architecture

IV. SYSTEM IMPLEMENTATION

The implementation follows a strict data flow pipeline to ensure accuracy and speed.

A. Station Registration (Admin)

When an Admin inputs a new station's details via the Admin Panel form, the frontend collects the station name, address, manually entered latitude and longitude, charging type, slot information, price, operating hours, and amenities. This data is submitted via a fetch() POST request to the /api/stations endpoint with a Bearer JWT token in the header. The backend verifies the token, checks the

 ©IJSRED: All Rights are Reserved

admin role via middleware, and saves the station document to MongoDB using Mongoose.

B. Proximity Querying (Haversine Algorithm)

When a user requests nearby stations, the frontend sends the user's current latitude, longitude, and search radius (in km) to the /api/stations/nearby endpoint. The backend retrieves all stations from MongoDB and applies the **Haversine formula** to calculate the great-circle distance between the user's coordinates and each station's stored coordinates. Stations within the specified radius are filtered, sorted by distance, and returned as a JSON array to the frontend.

The Haversine formula used is:

$a = \sin^2(Lat/2) + \cos(lat1) \times \cos(lat2) \times \sin^2(Lang/2)$

$c = 2 \times \text{atan2}(\sqrt{a}, \sqrt{(1-a)})$

distance = R × c (where R = 6371 km)

C. Real-Time Location Detection

The system uses the browser's HTML5 Geolocation API (navigator.geolocation.getCurrentPosition()) with enableHighAccuracy: true to acquire the user's GPS coordinates. The detected coordinates are displayed on the map with a blue circular marker, and a dashed green circle visualizes the selected search radius. Reverse geocoding via Nominatim displays the human-readable address in the popup.

D. Route Generation

Upon selecting a station and clicking "Get Directions," the Leaflet Routing Machine constructs a request URL containing the user's origin coordinates and the station's destination coordinates. OSRM processes this request and returns the route geometry. Leaflet decodes and renders this as a polyline on the map, along with turn-by-turn instructions in a fixed panel on the top-right of the screen.

E. Toast Notification System

A custom toast notification system is implemented in both map.js and admin.js using plain JavaScript DOM manipulation. Toast messages appear with animated slide-in/slide-out transitions and support four types: success, error, warning, and info. Each toast auto-dismisses after a configurable duration.

V. PERFORMANCE EVALUATION

The system was evaluated against a standard Google Maps implementation to assess viability.

Metric	**Google Maps API**	**Proposed System (OSM + OSRM)**	**Outcome**
Cost per 1000 Loads	~$7.00 USD	**$0.00 USD**	**100% Savings**
Geocoding Latency	~50ms	~120ms (Nominatim)	Slight delay, acceptable
Routing Speed	~100ms	**~40ms (OSRM)**	**Faster** (Contraction Hierarchies)
Data Freshness	Weekly Updates	**Real-time (Community Edited)**	Higher accuracy
Frontend Framework	SDK Required	**Vanilla JS (Zero dependency)**	**Lighter & Faster Load**

Analysis: While Nominatim showed slightly higher latency during address lookup compared to Google's Geocoding API, the OSRM routing engine outperformed the commercial alternative in raw route calculation speed. This is attributed to OSRM's C++ backend and Contraction Hierarchies algorithm. The use of Vanilla JS on the frontend eliminates framework overhead, resulting in faster initial page loads. The Haversine-based proximity query, while simpler than MongoDB's native geospatial operators, is fully sufficient for the scale of this application and avoids the need for specialized database indexing.

VI. OUTPUT AND ANALYSIS

 ©IJSRED: All Rights are Reserved

The system was tested with the following scenarios:

- **Location Detection:** The HTML5 Geolocation API successfully detected the user's current position with accuracy indicators displayed in the map popup.
- **Station Search:** The Haversine-based proximity search correctly returned and sorted nearby stations within the specified radius, visualized with a dashed circle overlay on the map.
- **Route Navigation:** The Leaflet Routing Machine successfully rendered turn-by-turn routes from the user's location to selected charging stations, with a fixed navigation panel displaying distance and duration.
- **Admin Operations:** The Admin Panel correctly enforced JWT-based role access, allowing only admin users to add, edit, or delete stations, with real-time table updates after each operation.

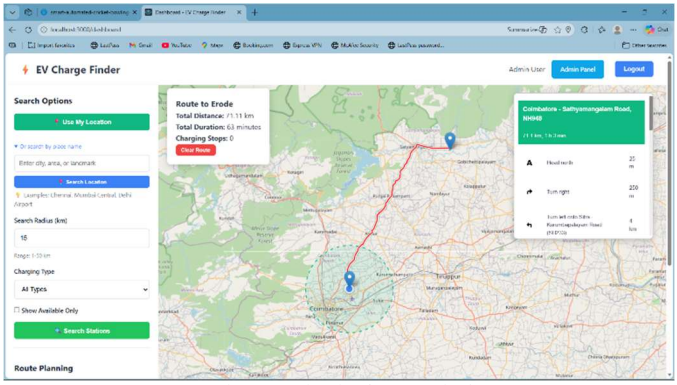

Fig. 2 Navigation/Route Display

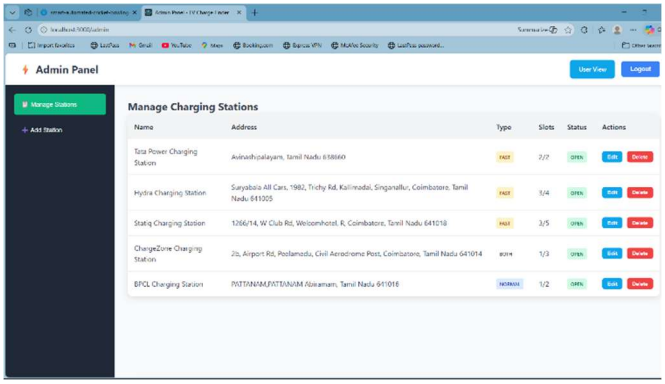

Fig. 3 Admin Panel - Manage Stations

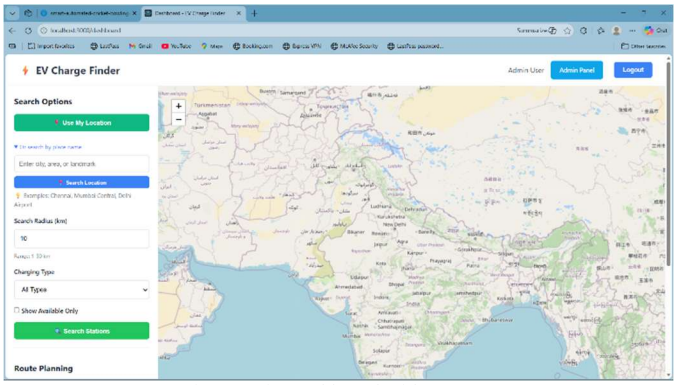

Fig. 4 Main Dashboard with Map

VII. FUTURE SCOPE

A. Offline Navigation & Edge Routing

Future work will integrate MBTiles to store localized map extracts on the client device and explore porting a lightweight routing engine to WebAssembly (Wasm) for offline functionality in areas with poor network connectivity.

B. AI-Driven Demand Forecasting

Future research will implement a Machine Learning (ML) module using Long Short-Term Memory (LSTM) networks to predict station availability based on historical session data (time-of-day, day-of-week, charging duration), replacing the current binary Open/Closed status.

C. MongoDB Geospatial Indexing

The current implementation uses a Haversine-based distance calculation in the application layer. A future enhancement will introduce MongoDB's native 2dsphere index and $near geospatial operator, offloading proximity calculations to the database engine for improved scalability at larger data volumes.

 ©IJSRED: All Rights are Reserved

D. Real-Time Slot Updates via WebSockets

The current slot availability is manually updated by admins. Future iterations will implement WebSocket (Socket.io) based real-time updates so that slot availability changes are pushed instantly to all connected users without requiring a page refresh.

E. Vehicle-to-Grid (V2G) Compatibility

As EV technology matures, the platform will be adapted to support Vehicle-to-Grid (V2G) transactions, effectively turning the station finder into a decentralized energy trading platform.

F. Community-Led Data Contribution

Leveraging the open nature of OpenStreetMap, a "Contributor Mode" will allow verified users to submit new station nodes directly

VIII. CONCLUSION

This research confirms that a Leaflet + OSM + OSRM stack combined with a Node.js + Express.js + MongoDB backend and a Vanilla JavaScript frontend is not merely a "free alternative" but a practical and performant architectural choice for EV infrastructure mapping. The system successfully implements JWT-based authentication, role-based access control, Haversine-based proximity search, real-time geolocation, Nominatim geocoding, and OSRM-powered turn-by-turn navigation — all without incurring any API licensing costs. By eliminating the pay-per-request model, the system democratizes access to navigation technology and proves that open-source software is the key to scalable, sustainable smart city solutions.

REFERENCES

[1]. Luxen, D., & Vetter, C. (2011). "Real-time routing with OpenStreetMap data." *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 513-516.

[2]. OpenStreetMap Foundation. (2024). "Nominatim: Search Engine for OpenStreetMap." [Online]. Available: https://nominatim.org

[3]. Leaflet.js Team. (2024). "Leaflet Routing Machine: A flexible routing plugin for Leaflet." [Online]. Available: https://www.liedman.net/leaflet-routing-machine/

[4]. Haklay, M. (2010). "How good is OpenStreetMap information? A comparative study of OpenStreetMap and Ordnance Survey datasets." *Environment and Planning B: Planning and Design*, 37(4), 682-703.

[5]. MongoDB Inc. (2024). "Geospatial Queries: $near vs $geoWithin." *MongoDB Official Documentation*.

[6]. Choudhary, P. (2023). "MERN Stack for Location-Based Services: A Performance Analysis." *International Journal of Computer Applications*, 185(4), 12-18.

[7]. Phuangsuwan, S., et al. (2024). "Economic Analysis of Commercial Location-Based Services in Mobile Applications." *Journal of Information Technology Management*, 16(2), 45-58.

[8]. Patel, R., & Sharma, A. (2025). "A Comparative Review of Open-Source GIS Frameworks for Web-Based Applications." *International Journal of Geoinformatics*, 21(1), 33-47.

[9]. Li, X., & Wang, Y. (2022). "Performance Challenges in Open-Source Map Rendering for Routing Applications." *Journal of Web Engineering*, 21(4), 1123-1145.

[10]. Mozilla Developer Network. (2024). "Geolocation API." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API

[11]. Express.js Team. (2024). "Express.js: Fast, unopinionated, minimalist web framework for Node.js." [Online]. Available: https://expressjs.com

 ©IJSRED: All Rights are Reserved