JOURNAL OF
Emerging Paradigms
in Computing Systems
JEPCS

Journal of Emerging Paradigms in Computing Systems (JEPCS) | Volume 02, Issue No. 01, 2026

ARTICLE TYPE: Research Article

Article ID: JEPCS-2026-0201-0060

Haazir Mechanic: Design and Functional Evaluation of a Location-Aware Mobile Platform for Roadside Vehicle Assistance

Zohaib Badar[1,*], Hafiz Faizan Ahmed[1]

[1] Faculty of Information Technology, University of Central Punjab, Lahore, Pakistan.

*Corresponding author: L1F20BSSE0548@ucp.edu.pk

Contributing authors: L1F20BSSE0548@ucp.edu.pk, L1F21BSSE0030@ucp.edu.pk

Received: October 20, 2025 **Accepted:** December 22, 2025

Abstract- Getting roadside assistance is difficult as the driver has to locate a mechanic, explain the problem, agree on the time and then follow up the service request through multiple calls or informal contacts. Haazir Mechanic was developed as a mobile prototype to integrate those steps in one, location-aware workflow. The application is implemented using Flutter for mobile client, Node.js and Express for REST services and MySQL for persistent data. GPS and mapping services can help you locate nearby mechanics. It also includes a troubleshooting assistant, although the paper refers to this as rule based rather than trained diagnostic model, despite the "AI chatbot" name used in parts of the interface. The only evidence available for evaluation is the report of the final year project. It reports 20 functional test cases across authentication, mechanic search, booking, troubleshooting and reviews. We detail five end-to-end cases. The aggregate summary includes six defects, all of which are marked corrected. The results demonstrate that the documented prototype workflows were successfully executed, but they do not demonstrate field reliability, usability with real drivers and mechanics, performance at production scale, or diagnostic accuracy. Thus, this article considers Haazir Mechanic as an applied software prototype. The architecture and service flow are rebuilt, the degree of intelligence implemented is clarified and Verified functional behavior is distinguished from capabilities still requiring security, scalability, provider verification, payment and field evaluation.

Keywords- Roadside assistance; location-based services; mobile application; mechanic discovery; rule-based troubleshooting.

1. Introduction

A vehicle break-down is a simple but urgent coordination problem. The driver needs to know who can help, where that help is, if the help is available, and how the request will be tracked once contact is made. Many of these steps are done by telephone, personal recommendation or separate service directories. This is more difficult to do and the driver is out of familiar territory.

Location based services can help to eliminate some of this uncertainty, by allowing a mobile application to combine the position of the driver with information on providers in the vicinity. In location-aware systems research, positioning, communication, evaluation, interface design, and privacy are not independent features, but related concerns [1]. The same idea is used in a more concrete setting in roadside assistance applications. For instance, Rescuennect combines GPS, mapping, provider information, and help requests into one mobile application [2]. Other recent systems also match vehicle owners and mechanics using location tracking and booking workflows [3, 4].

Table 1: *Core functions retained for the Haazir Mechanic evaluation.*

Function	Primary actor	Implemented purpose
Authentication	Vehicle owner / mechanic	Register a user, authenticate credentials, and restrict protected functions to an active session.
Mechanic discovery	Vehicle owner	Use location and service information to present relevant mechanic options.
Service booking	Vehicle owner / mechanic	Create a booking request, record the provider response, and maintain the request status.
Troubleshooting support	Vehicle owner	Return predefined guidance for common symptoms and direct unresolved cases toward mechanic assistance.
Status communication	Vehicle owner / mechanic	Report progress of a booking or service request and notify the relevant user.
Reviews and feedback	Vehicle owner	Store a rating or comment after a completed service interaction.
Administration	Administrator	Support user and mechanic oversight, service monitoring, and complaint handling as described in the project specification.

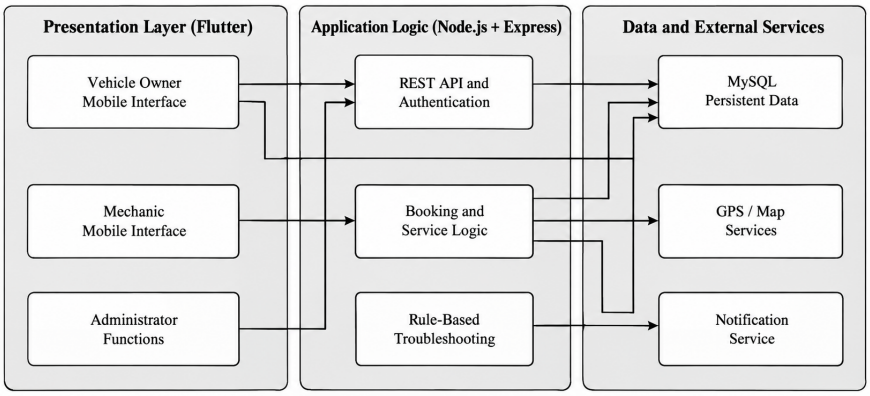


Figure 1: *Functional architecture reconstructed from the Haazir Mechanic project.*

The solution to this problem was to create Haazir Mechanic as a final year mobile software project. The prototype is an app that integrates mechanic discovery, service booking, status updates, troubleshooting help and post-service feedback. The user doesn't have to move from map to directory to another communication channel to start a request. Same system on provider side, a mechanic can get a booking and respond to it and update the service status.

The system has to be described more accurately, but the source project has a terminology problem that must be solved first. In several early studies, price estimation with machine learning has been discussed. It is also named "AI chatbot" from the mobile interface. The implementation record in the latter is more detailed. A learning model was not implemented due to the lack of a suitable labeled data-set within the project constraints. But the troubleshooting module has built-in guidance based on a rule-based knowledge base. What follows describes the implemented system, not the language previously proposed. no claimed diagnostic performance, machine learning accuracy or learned price prediction

Hence, the present paper studies Haazir Mechanic as a software system in the field. In this paper, we answer three questions:

- RQ1: What are the relationships of mechanic discovery, booking, service-status updates and troubleshooting support in the mobile architecture that has been implemented?
- RQ2: The functional tests documented demonstrate the main mechanic and vehicle-owner flows.
- RQ3: What are the technical and evaluation gaps before the prototype can be regarded as a production roadside-assistance service?

The contribution is intentionally limited. We use the project record to recreate the client-server architecture, describe the service life cycle in terms consistent with implementation, and present functional evidence without translating project requirements into measured results. That's an important distinction because the docs have requirements for responsiveness, concurrentity, but no benchmark logs for them. Security, payment and cross-platform operation is the same.

The rest of the paper is organized as follows. In Section 2, we place the prototype into the context of previous work on location-based and roadside-assistance systems. Section 3 describes the actors, architecture, data model, workflow and the implemented troubleshooting logic. Section 4 provides functional evidence for evaluation. Section 5 interprets the results and discusses the remaining engineering gaps. Section 6 highlights the limitation of the paper. Finally, Section 7 concludes the study.

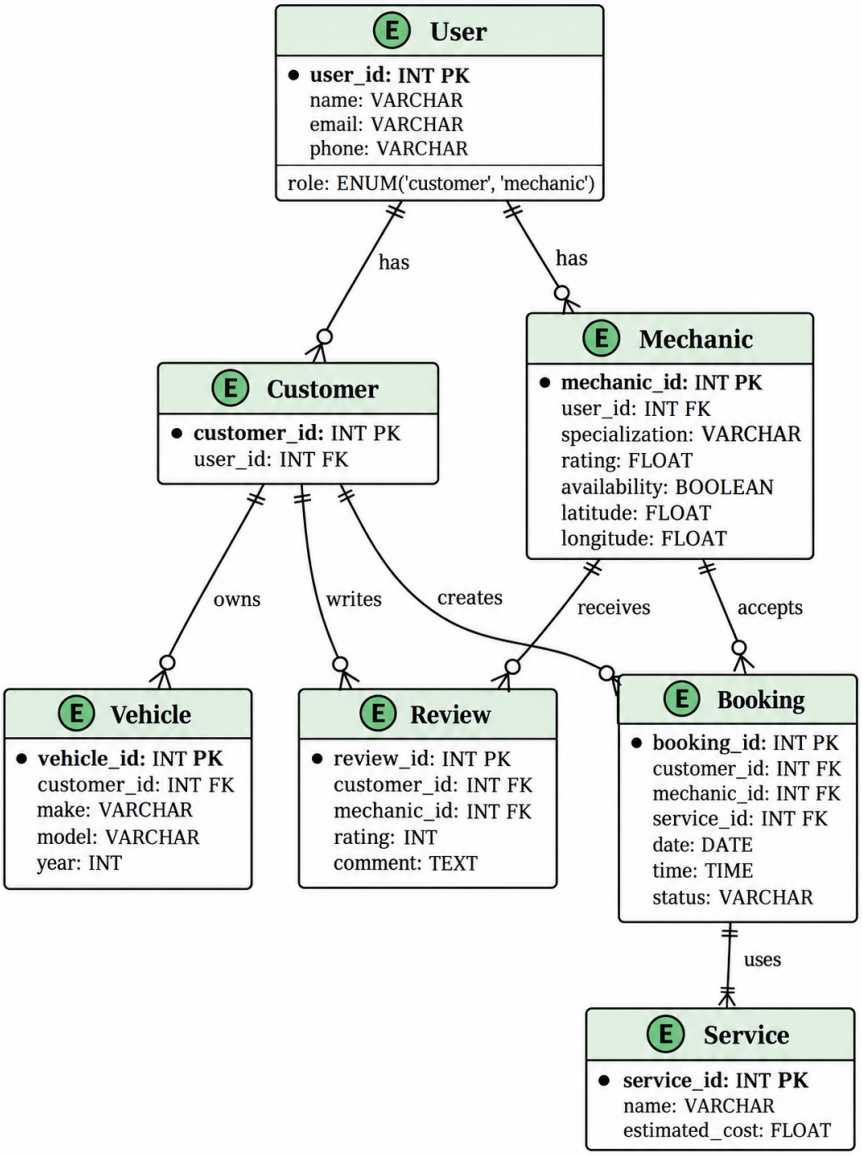


Figure 2: *Entity-relationship model supplied in the Haazir Mechanic project.*

2. Related Work

2.1. Location-Aware Service Design

Location-based services use a device or user position to change what information is presented or which service is offered. Huang et al. [1] describe the field in terms of positioning, modelling, communication, applications, interface design, evaluation, analysis of location data, and social implications. Roadside assistance touches several of these concerns at once. A wrong position can send a mechanic to the wrong place. Stale availability can produce a failed booking. Continuous location sharing also creates a privacy concern even when the interface itself works correctly.

Emergency-response research reaches a similar conclusion from a different direction. Audu et al. [5] use geographic information systems to support response to road-traffic incidents and show why spatial information is useful when resources must be directed to a physical event. Damaševičius et al. [6] discuss connected emergency services more broadly and emphasize timely data exchange between users, sensing or communication infrastructure, and response services. Haazir Mechanic is not an emergency management platform but the same coordination principle applies, in that the request must be tied to a place and to a provider that can act on it.

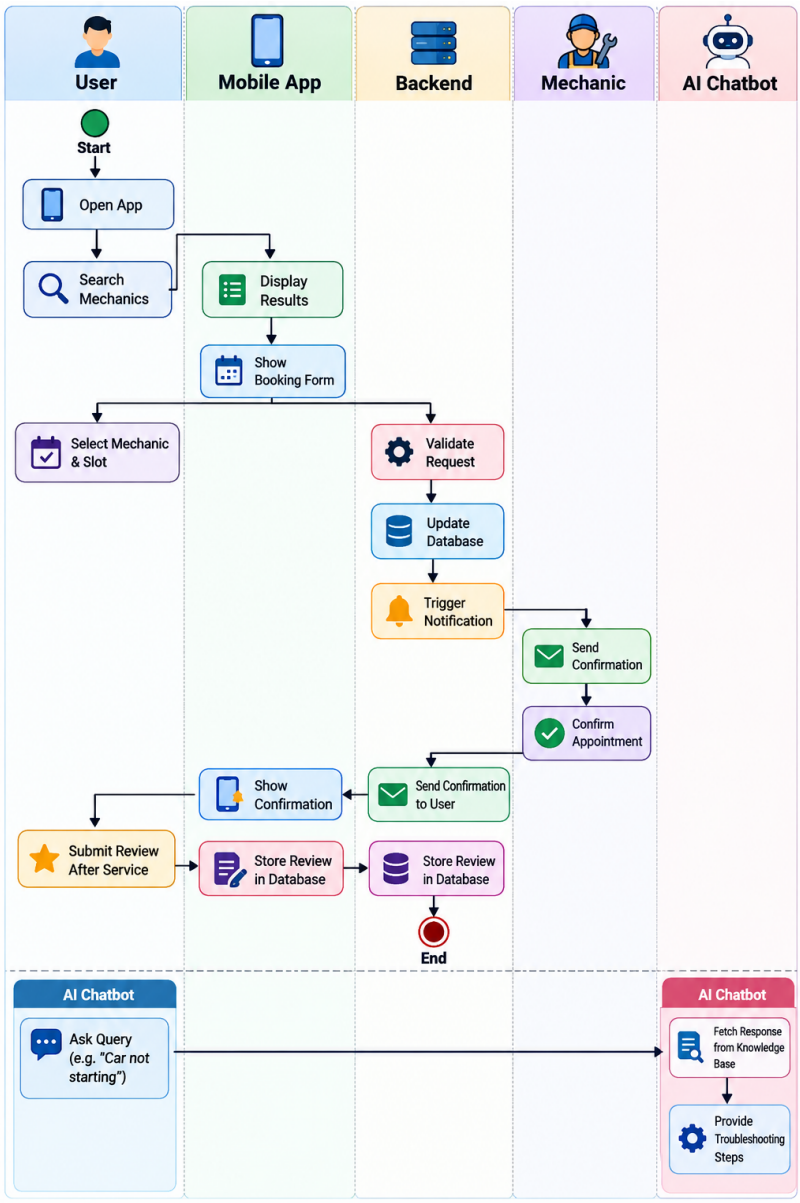


Figure 3: *Service-request and troubleshooting workflow reproduced from the project prototype.*

2.2. Mobile Roadside-Assistance Platforms

Mobile roadside assistance systems generally combine discovery, request creation and provider coordination. Padilla created the Rescuennect app that combines GPS and Google Maps with a list of roadside and emergency services [2]. The published evaluation used software quality criteria such as functionality and usability. The system also had provider tracking and multiple assistance categories, which is close to the service problem we are considering.

Aher et al. [3] developed an on-road vehicle breakdown assistance system intended to connect stranded motorists with registered mechanics, including support for searching mechanics in nearby locations. Thiagarajan et al. [4] later presented a mobile roadside assistance platform that combines real-time location information with service-provider discovery and related mobile service functions. These systems demonstrate that location is not the entire roadside workflow. A usable platform also requires a request state, provider response, communication, and some means of closing the service interaction.

Haazir Mechanic follows the same general service pattern, but the evidence for this article is different. The project record contains an architectural description, interface evidence, functional test cases, and a defect summary. Nor is there a controlled comparison with the cited systems or field measurements indicating faster roadside response. Thus, the design is not claimed to be better than the previous work but to be on position with it.

2.3. Technical Position of the Prototype

The implemented stack involves a conventional mobile client-server architecture. Flutter provides the cross-platform interface [7] Node.js and Express provide the server-side runtime and routing framework [8, 9] MySQL stores persistent application data [10] Map related interaction is implemented using Google Maps Platform [11], while Firebase Cloud Messaging is specified as the notification channel in the project architecture [12]. Token based routes are aligned with the JSON Web Token model defined in RFC 7519 [13].

These technologies are not offered as research contributions. Their importance is architectural: each is doing a different part of the service lifecycle. The mobile client collects a request with its location context. The backend does booking logic and updates state . The database has users, mechanics, services, bookings, vehicles and reviews. The core path is surrounded by map and notification services that communicate location and status information. The contribution of the project is in the way the pieces are assembled for the roadside workflow and how that workflow was tested.

3. System Design and Methodology

3.1. Actors and Functional Scope

The application has two main operation actors: a vehicle owner and a mechanic. The vehicle-owner record in the database diagram accompanying the project is labeled Customer. This paper uses vehicle owner in the prose to avoid changing terminology without explanation, and leaves Customer only in relation to the database entity as shown in Figure 2.

Vehicle owner can Register/Login, Share/Enter Location details, Search mechanics, Create a booking and check status, Use the troubleshooting assistant and Leave a review after service. Mechanic has a service profile and availability, receives bookings, accepts or declines requests and updates service status. The project specification also includes administrative functions, mainly for supervision of the user and the mechanic, but the documented functional tests are focused on the owner-mechanic workflow. Table 1 shows the functions used in the research evaluation. The wording is about real implemented behavior, not aspirational features mentioned elsewhere in the project report.

3.2. Architecture

The prototype features a three-part application structure. The flutter client handles screens, navigation, form input, validation and device access to location information [7]. Requests to change or retrieve application state are sent to the Node.js/Express backend using REST endpoints [8, 9]. Persistent records are stored in MySQL [10]. These core paths are supplemented by map and notification services, not replaced by them.

The architecture is shown in a form that makes those responsibilities explicit in Figure 1. The troubleshooting assistant is located in the application-logic layer because it currently acts in a rule-based manner. The project screenshots call it a "AI chatbot" but no trained model is reported.

3.3. Data Model

The project database distinguishes between shared account data and data for roles and transactions. A User record has common identity fields. A Customer record represents the vehicle owner, and a Mechanic record stores attributes that are specific to the mechanic, such as specialization, rating, availability and coordinates. Customers own the vehicles. A booking links a customer, a mechanic and a service. Reviews link a completed interaction back to the customer and mechanic involved.

The benefit of this separation is that a change in one part of the service does not have to be stored as free text within another record. For example, the mechanic's availability can change without changing the customer's vehicle record, and a review can be added without changing the original booking fields . Figure 2 shows the entity-relationship model.

3.4. Service Workflow

After authentication a normal service request begins . The vehicle owner browses the mechanics and the available time slots, selects a provider and makes an appointment. The back end checks the request and saves the state that results. The mechanic can accept or reject the booking. Once the request is acknowledged, more status changes are sent back to the user and the interaction can be closed with a review.

This booking path is different from the troubleshooting path. A user can enter a symptom such as "car not starting". The existing assistant does not infer a diagnosis from data. It retrieves pre-defined guidance from a knowledge base and can guide the user to mechanic assistance if the problem is not solved. Figure 3 presents the original project workflow. Its lane is labeled "AI Chatbot" as that wording appears in the source diagram; the implemented component is referred to as a rule-based troubleshooting assistant throughout this paper.

Table 2: *Technology stack identified in the project.*

Layer / service	Technology	Role in the prototype
Mobile frontend	Flutter / Dart	Screens, navigation, forms, client-side interaction, and access to location capabilities.
Backend	Node.js / Express	REST endpoints, authentication flow, booking logic, profile handling, validation, and service coordination.
Database	MySQL	Persistent storage for users, mechanics, vehicles, bookings, services, and reviews.
Location support	GPS / Google Maps	Device location, map display, and location-aware mechanic discovery.
Notifications	Firebase Cloud Messaging	Intended delivery of booking and service-status notifications.
Authentication	Email/password and token-based routes	Access control for user-specific and role-specific operations.
Troubleshooting	Rule-based knowledge base	Predefined responses to common vehicle symptoms and escalation toward mechanic assistance.

Table 3: *Representative functional test cases reported in the project.*

Test	Module	Expected behavior	Reported outcome
TC-1	Authentication	Valid credentials create a session and redirect the user to the dashboard.	Passed; actual result reported as expected.
TC-2	Mechanic search	Location, service type, and rating filters return a filtered mechanic list.	Passed; actual result reported as expected.
TC-3	Booking	A logged-in user selects an available mechanic, date, and time and creates a booking.	Passed; booking reported as created and visible.
TC-4	Troubleshooting assistant	The query “Car not starting” returns guidance or a mechanic recommendation.	Passed; actual result reported as expected.
TC-5	Reviews and feedback	A user submits a rating or comment for a completed service.	Passed; feedback reported as saved and linked to the mechanic profile.

3.5. Implemented Technology Stack

The project explicitly mentions the technologies listed in Table 2. They are presented as implementation choices, not proof of performance. The use of Flutter alone does not guaranty the same behavior on Android and iOS and the presence of MySQL does not guaranty performance under concurrent production traffic.

4. Functional Evaluation

4.1. Representative End-to-End Cases

The five test cases explicitly described in the project material are summarized in Table 3. They are functional in nature, ensuring that the expected state transition takes place when the user performs an action. They do not measure how fast or how reliably the same operation would behave under many simultaneous users.

4.2. Aggregate Test and Defect Log

The summary of the aggregate shows 20 cases. The tested modules reveal six defects, all of which are corrected as per the same summary. This distribution is reproduced in Table 4. Two defects each: authentication and booking. The mechanic search and the troubleshooting assistant each have one and the reviews module has none in the summary provided. The table supports a limited conclusion: the project team identified defects during functional testing and indicated that the listed defects were corrected before the final test summary was produced. It does not show the severity, recurrence, coverage percentage and behavior under the production load of the defect. Those details are not in the source material.

4.3. Screens Evidence

The screens of the prototype are another kind of evidence of implementation. Four interfaces extracted as shown in Figure 4. The first screen presents a choice of roadside service. The second asks for search parameters for nearby mechanics. The third is a mechanic and service pathway profile. The fourth is the troubleshooting interface.

The screens are aligned with the workflow described in Section 3, but screenshots are unable to validate the backend behavior. Their value here is limited, as they confirm the project did get an interactive mobile implementation, and was not stalled at the requirements or UML stage.section

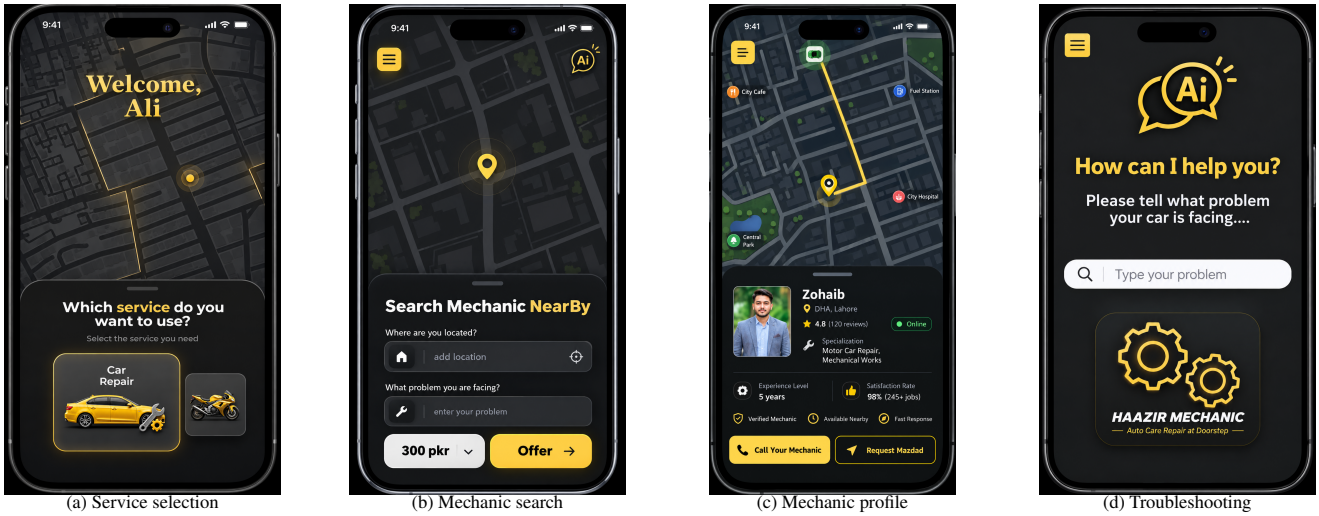
5. Results and Discussion

5.1. What the Functional Evidence Reveals

RQ2 has an answer at the prototype level, positive, in the project record. The expected result is shown in the five detailed cases. The aggregate summary includes 20 cases from five modules. The recorded defects are not concealed in the test summary. Six were identified and all six have been marked corrected. Together, these records indicate that the core flow was exercised beyond a static interface demo. There is strongest evidence for simple user actions . Authentication gives the session access to the required protected functions. Search applies the service and location inputs to filter the mechanic list. Booking creates a continuous service request. The troubleshooting assistant returns a pre-defined response for a

Table 4: *Project-reported test coverage and defect status.*

Module	Test range	Found	Corrected	Outstanding
User authentication	TC-01–TC-05	2	2	0
Mechanic search and filtering	TC-06–TC-10	1	1	0
Booking and scheduling	TC-11–TC-15	2	2	0
Troubleshooting assistant	TC-16–TC-18	1	1	0
Reviews and feedback	TC-19–TC-20	0	0	0
Complete tested set	TC-01–TC-20	6	6	0



(a) Service selection (b) Mechanic search (c) Mechanic profile (d) Troubleshooting

Figure 4: *Representative Haazir Mechanic interfaces.*

documented symptom . Service feedback is saved. This is what the paper can definitely talk about, because it corresponds to explicit test records. Some of the broader claims are not supported by the evidence. A booking test is not proof that the backend can scale to low latency with hundreds of concurrent users. A successful case of mechanic-search does not confirm the correctness of location in dense or poorly mapped regions. A rule-based response to "car not starting" is not a diagnosis. Thus the paper differentiates between functional completion and operational quality.

5.2. Integrating Architecture and Workflow

RQ1 concerns integration, not novelty of individual technologies. The mobile client gathers user input and location context, but the booking state is managed outside the interface. The backend authenticates the request and handles it . Mysql keeps persistent data . Mapping and notifications support flow around that core path. This separation can be seen in the architecture of the project and the ordering of the test cases. The design is modular enough that a single component can change without having to rewrite every screen. For example, the troubleshooting assistant could later be replaced with a validated diagnostic service without impacting the booking and review data model. The same applies to the notifications. This is an architectural choice, not evidence that those replacements have been implemented or validated.

5.3. Comparison to Related Systems

Haazir Mechanic isn't the first mobile system to use GPS for roadside help. Rescuennect already incorporates mapping, provider information, and assistance requests [2]. Similar to the systems proposed by Sharmi and Logaiyan [3] and Kannan et al. [4], the systems described here also use location-aware matching and mechanic-service workflows. Thus, the difference relevant to this paper is not that the service concept is new, but the documented implementation and test record of the Haazir Mechanic prototype. The prototype is not as complex as systems that use telematics, diagnostic models, dynamic pricing, or learned provider ratings. The current version is simple and therefore easy to describe and reproduce. This simplicity also limits what can be said about it. The troubleshooting module works according to a set of predefined rules. The implemented evaluation does not include any price estimation. The provider ranking is not a learned model. Any future addition of those capabilities should be treated as a separate research component with its own dataset, baselines and error analysis.subsection

6. Limitations

- The first limitation is the provenance of the results. The raw automated test logs were not included with the paper, so the results could not be independently replayed in this paper.
- The second limitation is the evaluation type. The test suite checks the functional behavior. It does not involve a controlled usability study with stranded drivers or mechanics, a timed field deployment, network-failure experiments, or production-scale load testing. Response time and availability continue to be requirements as the report does not include benchmark traces for them.
- A third limitation refers to the internal consistency of the source manual. Some status tables call modules planned or incomplete, but later test summaries and prototype screens refer to those functions as implemented. That doesn't mean that all the planned features were equally completed.
- Finally, the project does not support the early numerical claims of machine-learning cost estimation or reduced booking time. The learning component was not integrated, as evidenced by the later record, so these values are not included. There is no supporting dataset or timing experiment. Removing unsupported numbers reduces the paper, but does make the remaining claims replicable from the evidence at hand.

7. Conclusions and Future Directions

Haazir Mechanic is a Flutter, Node.js/Express, and MySQL application featuring location-aware mechanic discovery, booking, service-status coordination, rule-based troubleshooting, and post-service feedback. The project record reports 20 functional test cases in 5 modules. The five end-to-end cases are detailed and aggregate summary records six defects, all marked corrected.

These results support a modest conclusion. The prototype implements and tests the main workflow for roadside assistance as documented by the project team. They do not demonstrate production readiness, large scale reliability, field usability, diagnostic accuracy or economic benefit. The paper does not contain the implementation or experiment to back up the earlier claims of machine learning and booking time.

Future work will include backend testing as it is required due to concurrent load, intermittent connectivity and failure recovery.

Author Declarations

Author Contributions (CRediT): Zohaib Badar: Conceptualization, Software, Investigation; Hafiz Faizan Ahmed: Validation, Visualization, and Writing - Original Draft. Both authors contributed to the development and documentation of the Haazir Mechanic final-year project.

Funding: This work was completed as an undergraduate final-year project. No external funding is reported in the submitted project documentation.

Data Availability Statement: No research dataset was generated for this manuscript.

Acknowledgments: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Ethics Statement: Not applicable.

Declaration of Generative AI: An AI-assisted tool was used during manuscript preparation for language editing, restructuring of the final-year project documentation into a journal-paper format. The authors are responsible for verifying all technical claims, references, figures, results, and declarations before publication.

References

[1] H. Huang, G. Gartner, J. M. Krisp, M. Raubal, and N. Van de Weghe, "Location based services: Ongoing evolution and research agenda," *Journal of Location Based Services*, vol. 12, no. 2, pp. 63–93, 2018.

[2] A. T. Padilla, "An in-depth analysis of the mobile application "rescuennect" leveraging telematics technology for roadside assistance and emergency response solutions," *International Journal of Soft Computing and Engineering*, vol. 14, no. 5, pp. 1–10, 2024.

[3] S. S. Aher, V. T. Unhale, P. B. Gade, and T. D. Patil, "On road vehicle breakdown assistance," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 11, no. 11, pp. 85–88, 2022.

[4] G. Thiagarajan, B. Santhosam, S. R. David, K. A. Deenadhayal, and A. J. Nathan, "Vehicle breakdown assistance system for on-road," *International Research Journal on Advanced Engineering Hub*, vol. 3, no. 4, pp. 1781–1785, 2025.

[5] A. A. Audu, O. F. Iyiola, A. A. Popoola, B. M. Adeleye, S. Medayese, C. Mosima, and N. Blamah, "The application of geographic information system as an intelligent system towards emergency responses in road traffic accident in ibadan," *Journal of Transport and Supply Chain Management*, vol. 15, p. a546, 2021.

[6] R. Damaševičius, N. Bacanin, and S. Misra, "From sensors to safety: Internet of emergency services (IoES) for emergency response and disaster management," *Journal of Sensor and Actuator Networks*, vol. 12, no. 3, p. 41, 2023.

[7] Y. Cheon and C. Chavez, "Converting android native apps to flutter cross-platform apps," in *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2021.

[8] Node.js Project, "Node.js 22.0.0," April 2024, [Online]. Available: https://nodejs.org/en/blog/release/v22.0.0.

[9] W. Todd and Express Technical Committee, "Introducing express v5: A new era for the node.js framework," October 2024, [Online]. Available: https://expressjs.com/en/blog/2024-10-15-v5-release/.

[10] Oracle, "MySQL 8.4 release notes: Changes in MySQL 8.4.0," April 2024, [Online]. Available: https://dev.mysql.com/doc/relnotes/mysql/8.4/en/news-8-4-0.html.

[11] A. A. Ahmed, "A privacy-preserving mobile location-based advertising system for small businesses," *Engineering Reports*, vol. 3, no. 11, p. e12416, 2021.

[12] G. Albertengo, F. G. Debele, W. Hassan, and D. Stramandino, "On the performance of web services, google cloud messaging and firebase cloud messaging," *Digital Communications and Networks*, vol. 6, no. 1, pp. 31–37, 2020.

[13] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," Internet Engineering Task Force, RFC 7519, May 2015.