



iJRASET

International Journal For Research in
Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14

Issue: X

Month of publication: October 2026

DOI:

www.ijraset.com

Call:  08813907089

E-mail ID: ijraset@gmail.com

FuelWise: A MERN-Stack Web Application for Fuel Consumption and Expense Tracking

Bharath Kumar N¹, Gokul R², Godlin Mejo K³, Xavier Mary A⁴

Department of Computer Science and Engineering, Adhiyamaan College of Engineering (Autonomous), Dr. M.G.R. Nagar, Hosur, Tamil Nadu, India

Abstract: Vehicle owners often keep refuelling details in notebooks, receipts or scattered spreadsheets, which makes fuel use and spending hard to review. This paper presents FuelWise, a web application that records refuelling events and summarises them on a single dashboard. FuelWise follows a three-tier architecture: a React 19 single-page front end, a Node.js/Express 5 REST API and a MongoDB database accessed through Mongoose. A user can add, edit, delete and search records of vehicle name, odometer reading, fuel quantity, fuel cost and date, and the interface computes total fuel, total expense, record count and monthly expense for the six most recent months. The paper describes the design, data model, implementation and a test plan. Manual checks with sample data confirmed the totals and search behaviour. The mileage and distance columns, however, read fields that the evaluated back end neither stores nor calculates, and the system lacks authentication and server-side validation; these limitations and the planned enhancements are discussed. FuelWise is an academic prototype, not a production system.

Keywords—fuel tracking, mileage, vehicle expense management, MERN stack, React, Node.js, Express, MongoDB, REST API.

I. INTRODUCTION

Fuel is one of the largest recurring costs of owning a vehicle, and understanding consumption requires a record of each refuelling: the date, the quantity bought, the amount paid and the odometer reading. When these details are kept on paper, in message threads or in unrelated spreadsheets they are easily lost, mistyped or never summarised, and totals must be calculated by hand.

This paper presents FuelWise, a web application prototype that brings these activities into one interface. It lets a user record refuelling entries, search them by vehicle name, edit or delete them, and view summary indicators and a month-wise expense chart. The objectives are: (i) to capture the five fields needed for later analysis; (ii) to validate input in the browser and report errors clearly; (iii) to store records persistently through a REST API; and (iv) to derive totals and monthly trends automatically.

The scope is limited to the implemented functionality: a single-page dashboard, a record form, a searchable history table, summary cards, a monthly chart and four CRUD endpoints. User accounts, multi-user data separation, server-side mileage calculation, receipt upload and data export are outside the evaluated implementation.

II. RELATED CONCEPTS

Fuel economy is commonly expressed as distance per unit of fuel; a frequently described manual method is to fill the tank completely, note the odometer and the quantity added, and divide the distance since the previous fill-up by that quantity [1]. FuelWise stores the quantities such a calculation needs, but the evaluated server does not perform it.

A web information system is typically organised into presentation, application and data tiers [2]. In FuelWise these are a React interface [3], an Express server [4] and a MongoDB document database [5]. Representational State Transfer maps HTTP methods to create, read, update and delete operations on resources identified by URLs [6], and Mongoose supplies schemas and model methods for MongoDB [7]. Dashboards that present a few totals and a trend chart help users notice changes that a raw list hides.

III. SYSTEM ANALYSIS AND REQUIREMENTS

A. Existing Workflow and Proposed System

In a manual workflow, refuelling details are written in a notebook, kept as receipts, noted in a messaging application or entered irregularly into a spreadsheet. Odometer readings are forgotten, entries for different vehicles are mixed, and totals must be calculated by hand. These are general risks of unstructured record keeping, not findings of a field study. A spreadsheet can address several of them, but the user must design the columns, write the formulas and maintain the file on one device. FuelWise instead offers a ready-made form, built-in validation, central storage and automatic summaries.

B. Functional Requirements

The system shall: (FR-1) load all fuel records when the dashboard opens; (FR-2) accept and store a record of vehicle name, odometer, fuel litres, fuel cost and date; (FR-3) reject an empty vehicle name, a negative odometer reading or cost, a non-positive fuel quantity and a missing date before sending a request; (FR-4) edit and (FR-5) delete a record after confirmation; (FR-6) filter the history table by vehicle name; (FR-7) display total fuel, total expense and the record count; (FR-8) display monthly expense for the six most recent months; (FR-9) display average mileage from the mileage values greater than zero; and (FR-10) report an error when the back end or database is unreachable.

C. Non-Functional Requirements

Usability: labelled fields, placeholders and clear success and error messages. **Performance:** acceptable response for a personal dataset, since all records are processed in the browser. **Reliability:** an error message when the back end is unavailable; production use would need monitoring and backups. **Security:** the prototype does not meet a production baseline (no authentication, open CORS). **Maintainability:** the back end separates server, model and routes, whereas the front end keeps all logic in one component.

D. Database Requirements

FuelWise stores records in MongoDB through the Mongoose model FuelRecord. By Mongoose's default naming rule the collection is named fuelrecords; this should be confirmed against the actual database, since no database export was available. The database name and credentials are supplied through the MONGO_URI environment variable and must be kept out of version control. The history table and average-mileage card read fields named mileage and distance, and the route handler sorts by createdAt, but none of these is defined in the schema, and timestamps are not enabled; unless they exist in another version of the code or database, they are absent from the returned documents. The database should be protected with authentication and network restrictions, and server-side validation should enforce sensible limits for odometer, quantity and cost.

IV. SYSTEM DESIGN

A. Architecture

Fig. 1 shows the architecture. A single React App component holds the interface state, renders the form, summary cards, chart and history table, and uses Axios to call the API. The base URL is read from the REACT_APP_API_URL environment variable and defaults to http://localhost:5000/api/fuel. The Express server enables CORS, parses JSON bodies, connects to MongoDB using the MONGO_URI environment variable and mounts one route group under /api/fuel. The four endpoints are listed in Table I.

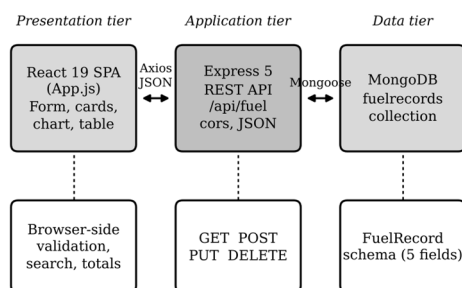


Fig. 1 High-level architecture of FuelWise

TABLE I REST ENDPOINTS OF THE BACK END

Method	Path	Operation
GET	/api/fuel	List all fuel records
POST	/api/fuel	Create a record
PUT	/api/fuel/:id	Update a record by identifier
DELETE	/api/fuel/:id	Delete a record by identifier

B. Data Model

Each refuelling event is stored as one document of the FuelRecord model (Table II). Mongoose adds a unique identifier and a version key. The schema declares types and required flags only; no minimum or maximum limits are defined.

TABLE II DATA DICTIONARY OF THE FUELRECORD SCHEMA

Field	Type	Description
vehicleName	String	Vehicle name (required)
odometer	Number	Odometer reading in km (required)
fuelLiters	Number	Fuel added, in litres (required)
fuelCost	Number	Amount paid, in rupees (required)
date	Date	Refuelling date (default: now)

C. Functional Modules

Record entry. The form collects the five fields; the date defaults to the current date. On submission the vehicle name is trimmed, numeric strings are converted to numbers and a POST request is sent; the form is then reset and the list reloaded. **Record maintenance.** Each table row has edit and delete actions. Editing loads the record into the same form and sends a PUT request; deleting asks for confirmation before sending a DELETE request. **Dashboard.** Four cards show total fuel, total expense (formatted in Indian rupees), average mileage and the number of records. **Monthly chart.** Fuel cost is grouped by month and year, the last six groups are kept and each bar is scaled to the largest monthly total. **History and search.** A search box filters the table by case-insensitive substring match on the vehicle name, without a server request.

D. Use Cases and Workflow

The only actor is the vehicle owner; there is no administrator role. The workflows are: *view dashboard* (all records are requested and the cards, chart and table are shown, or an error advises starting the server and checking MongoDB); *add record* (invalid input produces an error, valid input is stored and the totals update); *edit record*; *delete record*; and *search history*. Fig. 2 shows how a submitted record flows through validation, storage and the dashboard refresh.

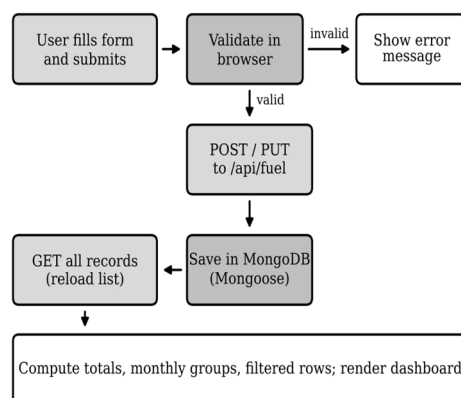


Fig. 2 Record entry and dashboard refresh workflow

V. IMPLEMENTATION

The project has a client folder (Create React App [9], React 19, Axios [10]) and a server folder (Express 5, Mongoose 9, cors, dotenv, nodemon). The API listens on port 5000. All interface state is kept in React hooks, and totals and monthly groups are recomputed with memoised calculations only when the record list changes.

TABLE III TECHNOLOGY STACK DECLARED IN THE PROJECT

Tier	Component	Role
Front end	React / React DOM 19	Component-based interface, state hooks
Front end	react-scripts 5.0.1	Development server and build tooling
Front end	Axios 1.x	HTTP requests to the API
Back end	Node.js, Express 5	Runtime, routing, JSON parsing
Back end	Mongoose 9	Schema and MongoDB access
Back end	cors, dotenv, nodemon	CORS, environment variables, auto-restart
Database	MongoDB	Persistent storage of records

Bootstrap, react-icons and Recharts are declared as dependencies but are not imported by the inspected source; the monthly chart is built from plain HTML elements. Any desktop operating system with a current browser, Node.js and a local or hosted MongoDB server can run the application; a dual-core processor with 4 GB of memory is sufficient for development.

Software description. React builds the single-page interface and manages state through the useState, useEffect, useCallback and useMemo hooks. Axios provides a promise-based JSON client between browser and server. Create React App supplies the development server, production build and Jest-based test runner. Node.js lets the front end and back end share one language, Express handles routing and middleware, cors adds cross-origin headers and dotenv loads the connection string. Mongoose provides the schema, the model and the query methods find, save, findByIdAndUpdate and findByIdAndDelete. Plain CSS provides the layout, with media rules at 800 px and 480 px that collapse the grids on narrow screens.

A. Client-Side Validation

The form is a controlled form that applies the rules of Table IV before building the request. These checks run only in the browser; a client that calls the API directly bypasses them, and the server enforces only Mongoose type casting and required fields.

TABLE IV CLIENT-SIDE VALIDATION RULES

Field	Rule applied before sending
Vehicle name	Not empty after trimming; max. length 80
Odometer	Present and not negative
Fuel filled	Present and greater than zero
Fuel cost	Present and not negative
Date	Present; defaults to current date

B. Dashboard Calculations

Total fuel and total expense are sums of fuelLiters and fuelCost over all loaded records, and the record count is the array length. Average mileage is the mean of the mileage field over records where it is greater than zero (zero if none). Currency is formatted with the Intl.NumberFormat API using the en-IN locale [8]. All records are fetched at once and processed in the browser, which suits a personal dataset but would need pagination for large ones.

VI. TESTING AND RESULTS

A test plan of twenty cases covers loading, adding, validating, editing, deleting and searching records, dashboard calculations, error handling, persistence and layout. No executed test log existed for the prototype, so the cases were exercised as follows. The client was compiled with react-scripts and run in a browser against a temporary in-memory stand-in for the API that mimics the four endpoints; MongoDB was not used, and the records shown are sample data. Table V lists twelve of the twenty cases with their status; the remaining cases are proposed specifications that were not executed.

TABLE V SELECTED TEST CASES AND EXECUTION STATUS

ID	Test case	Status
TC-01	Load dashboard with no records	Observed (stand-in API)
TC-02	Add a valid fuel record	Observed (stand-in API)
TC-03	Whitespace-only vehicle name	Observed (stand-in API)
TC-06	Edit an existing record	Partly observed
TC-08	Delete a record and confirm	Not executed
TC-10	Search by part of vehicle name	Observed (stand-in API)
TC-12	Totals and record count	Observed: 116.50 L, ₹12,013.00, 8
TC-13	Monthly grouping, six-month limit	Partly observed
TC-14	Average mileage without mileage field	Observed: 0.00 km/L
TC-15	Back end or database unavailable	Not executed
TC-18	Persistence after restart (MongoDB)	Not executed
TC-20	Default Create React App test	Executed: failed

With eight sample records the dashboard showed 116.50 L of fuel, ₹12,013.00 of expense and a record count of eight, matching the manual sums, and the monthly summary grouped the records into three months. Searching by part of a vehicle name returned only matching rows, and an empty vehicle name produced the message “Please enter a vehicle name.” without sending a request. The only automated test is the default Create React App example, which fails because the interface no longer contains the text it searches for. These results demonstrate the behaviour of the client code; they do not establish the behaviour of the real Express/MongoDB back end, which remains to be tested.

VII. DISCUSSION

Table VI relates each need that motivated FuelWise to what the prototype provides and to the gap that remains. FuelWise fits best an individual owner or a small number of vehicles with a modest number of records. It fits less well multi-user fleets, privacy-sensitive deployments, offline use and large datasets, because those capabilities are not implemented or verified.

TABLE VI PROBLEM-SOLUTION FIT

Need	Provided	Remaining gap
Record refuelling	Entry form and POST endpoint	Range checks only in browser
Maintain records	Search, edit and delete	No structured error replies
Summarise use	Totals, count, monthly chart	Mileage and distance show zero
Protect data	Environment-based DB string	No login or authorisation

The main threat to the validity of the results is that they were obtained against a stand-in API and sample data, not a MongoDB-backed deployment with real drivers. Usability has not been evaluated with users, no performance measurements exist, and cross-browser and responsive behaviour were not tested. The reported agreement between displayed and manual totals therefore confirms only the client-side arithmetic.

VIII. LIMITATIONS AND FUTURE SCOPE

The evaluated prototype has the following limitations (the default date also uses the UTC date, which can differ from the local date shortly after midnight in time zones ahead of UTC):

- Distance, per-record mileage and average mileage are read from fields that the schema does not define and the server does not calculate, so they are displayed as zero.
- The API has no authentication or authorisation, CORS is open to all origins, and every client can read and change all records.
- Server-side validation is limited to types and required flags; route handlers have no try/catch blocks, and a PUT for an unknown identifier returns null with a success status.
- The GET route sorts by a field (createdAt) that the schema does not define, so the monthly chart depends on the order of the returned records; several class names used in the interface have no style rules.

Future work includes server-side mileage calculation from the previous odometer reading of the same vehicle, validation with value limits and consistent error responses, user registration with token-based authentication and per-user data, a vehicle master list, pagination, date filters and CSV export, automated component and API tests, and usability trials with real drivers.

IX. CONCLUSION

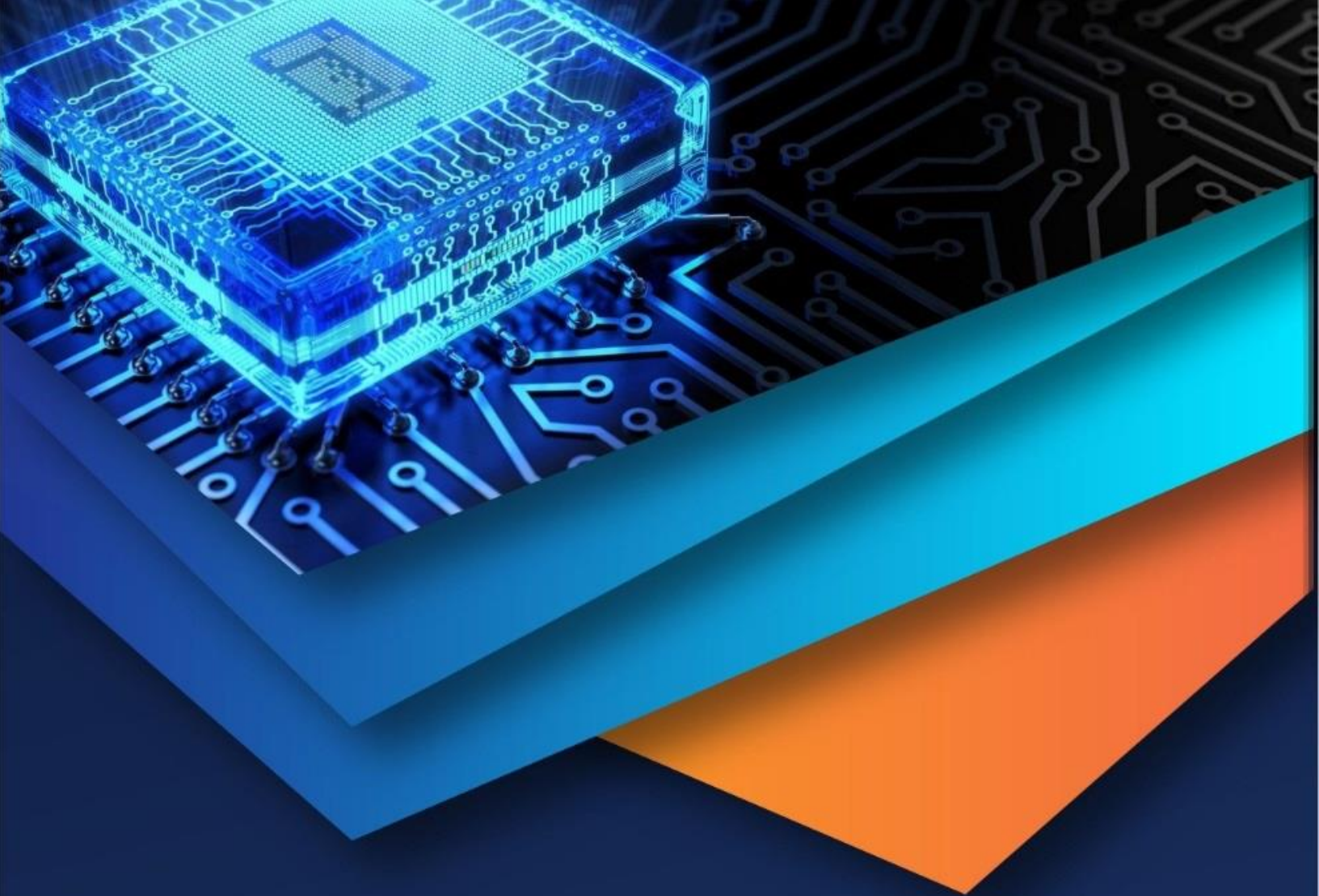
FuelWise demonstrates a lightweight, three-tier approach to recording vehicle refuelling data, storing it in MongoDB and presenting totals and monthly expense on a dashboard. The prototype supports a complete create, read, update, delete and search workflow, and the sample-data checks agree with manual calculations. It should be treated as an academic prototype: the mileage analysis suggested by its interface is not yet produced by the back end, and security and validation must be strengthened before it is used with real personal data.

X. ACKNOWLEDGMENT

The authors thank their supervisor, Ms. A. Xavier Mary, Assistant Professor, and Dr. G. Fathima, Professor and Head, Department of Computer Science and Engineering, Adhiyamaan College of Engineering, Hosur, for their guidance and encouragement.

REFERENCES

- [1] U.S. Department of Energy, "Fuel economy." [Online]. Available: <https://www.fueleconomy.gov/>
- [2] I. Sommerville, *Software Engineering*, 10th ed. Harlow, U.K.: Pearson, 2015.
- [3] Meta Open Source, "React documentation." [Online]. Available: <https://react.dev/>
- [4] OpenJS Foundation, "Express 5.x API reference." [Online]. Available: <https://expressjs.com/>
- [5] MongoDB, Inc., "MongoDB manual." [Online]. Available: <https://www.mongodb.com/docs/manual/>
- [6] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. of California, Irvine, CA, USA, 2000.
- [7] Mongoose Contributors, "Mongoose documentation." [Online]. Available: <https://mongoosejs.com/docs/>
- [8] MDN Web Docs, "Intl.NumberFormat," Mozilla. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Intl/NumberFormat
- [9] Create React App Contributors, "Create React App documentation." [Online]. Available: <https://create-react-app.dev/>
- [10] Axios Contributors, "Axios documentation." [Online]. Available: <https://axios-http.com/>



10.22214/IJRASET



45.98



IMPACT FACTOR:
7.129



IMPACT FACTOR:
7.429



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089  (24*7 Support on Whatsapp)