International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

RideMate: A Smart Campus Ride-Sharing Platform for Efficient, Secure and Sustainable Student Transportation

Prakash Das[1], Prajwal Pal[2], Shreya Kulkarni[3], Sharvari Deshpande[4], Vedant Bhoyar[5], Neel Rehpade[6]

Prof. Surabhi Chavhan[7],

[1,2,3,4,5,6] B.tech CSE G H Raisoni University, Amravati

[7]Guide, CSE G H Raisoni University, Amravati

Abstract—Students who commute to the same campus often travel in parallel without knowing it: one rider has an empty pillion seat, while a classmate on the same road pays for an auto-rickshaw. Informal arrangements made in messaging groups seldom cover seat approval, safety, or feedback. This paper presents RideMate, a web-based ride-sharing platform built for a single campus community. Drivers publish rides with a route, time, seat count and vehicle number; passengers search, request a seat, and wait for the driver to accept or reject the request. Once accepted, a passenger can follow the trip on an OpenStreetMap-based map using the browser's geolocation, raise an SOS alert that captures the current position and ride details, and, on arrival, rate the ride on a five-star scale and optionally contribute to charity through a QR-code page. The application is written in Python with Flask, was prototyped on SQLite, and was later moved to PostgreSQL hosted on Supabase, with the Flask service deployed on Render. We describe the design, the database schema, the booking state machine and the implementation, and report functional testing carried out with driver and passenger accounts. Load testing and a field study with real commuters are left for future work.

Key Words: ride sharing, carpooling, smart campus, Flask, PostgreSQL, GPS tracking, rider safety, web application

I. INTRODUCTION

A campus is a place where hundreds of people leave and arrive at roughly the same hours, usually along a small number of roads. Many of them come on their own two-wheelers or cars, and a good share of those vehicles carry only the driver. Meanwhile, students without a vehicle depend on shared autos, buses or whatever a friend can offer. The unused seat and the person who needs it are often a few hundred metres apart and never meet.

Ride sharing is an established answer to this mismatch, and the literature on it is large [1] – [4]. Most of that work, however, concerns city-wide services with paid drivers or open membership. A campus is different in three ways that shaped our design. The user base is small and mostly known to each other through classes and departments, so trust can be anchored to a college identity rather than a payment history. Routes repeat (campus to a metro station, a railway station, a residential area), so a simple list of offered rides is often more useful than a complex matching engine. And the people at risk are mostly young commuters travelling in the evening, which makes safety features a core requirement and not an add-on. This paper describes RideMate, a web application that connects student drivers with student passengers on the same campus. It was developed as a final-year project, first as a local prototype and then as a cloud-hosted multi-user system. The contributions are as follows:

• A complete ride lifecycle in one application: offer, search, request, driver approval, tracking, completion, rating and optional charity.

• A driver-approval booking model with explicit Pending, Accepted and Rejected states, seat accounting and notifications.

• Browser-based GPS tracking on an OpenStreetMap map with destination geocoding and routing, together with an SOS flow tied to an emergency contact stored in the user profile.

• A description of the move from a SQLite prototype to a PostgreSQL (Supabase) back end with the Flask service hosted on Render, and the reasons for it.

The rest of the paper is organised as follows. Section II reviews related work. Section III states the problem and objectives. Sections IV and V present the architecture and database design. Section VI describes the implementation with screenshots of the running system, Section VII covers deployment, Section VIII reports testing, and Sections IX–XI discuss limitations, future work and conclusions.

II. RELATED WORK

Furuhata et al. survey ridesharing systems and classify them by matching style, participants and objectives; they also list recurring obstacles such as trust between strangers, schedule flexibility and the critical mass needed for a service to be useful [1]. Agatz et al. review optimisation methods for dynamic ridesharing, where riders and drivers are matched in real time to reduce total travel [2]. At a larger scale, Alonso-Mora et al. show how a fleet of shared vehicles can be assigned to requests dynamically, using New York taxi data to estimate how few vehicles would be needed [5]. These approaches concentrate on the quality of the match.

A second line of work looks at ridesharing as a social and policy question. Chan and Shaheen trace the history of ridesharing in North America and the factors that helped or held back adoption [3], and Shaheen and Cohen later clarify the terminology around shared and pooled rides and discuss their possible impacts [4]. Both stress that user acceptance depends on convenience and perceived safety, which is the angle Ride Mate takes.

On the technology side, RideMate relies on open geographic data. OpenStreetMap is a volunteer-built map database [6], Leaflet renders it in the browser [7], OSRM computes routes over it [8], and Photon provides geocoding [9]. Positioning is obtained from the W3C Geolocation API supported by current browsers [10].

Compared with the systems above, RideMate does not try to optimise matching. It targets a closed campus community and puts weight on four practical things: an explicit driver-approval step, visible booking status and notifications, in-ride tracking with an SOS option, and a simple post-ride rating.

III. PROBLEM STATEMENT AND OBJECTIVES

The problems that motivated the project were observed in everyday campus travel: students find it hard to get affordable transport at the right time; empty seats in private vehicles go unused; arrangements are made by phone or chat with no record of who agreed to what; there is no way to follow a ride or call for help from within the arrangement; and no structured feedback exists about drivers. The problem addressed here is therefore stated as follows.

To design and implement a secure, responsive, campus- specific ride-sharing platform that connects student drivers and passengers, manages booking and approval, provides GPS- based ride tracking and emergency assistance, and collects post-ride feedback.

From this statement the following objectives were derived: (1) let drivers publish rides with route, date, time, seats and vehicle; (2) let passengers browse rides and request a seat; (3) let drivers accept or reject requests while the system keeps seat counts consistent; (4) notify both sides of status changes; (5) provide map-based tracking after acceptance; (6) provide an SOS mechanism during an active ride; (7) store a one-to-five rating per completed booking; and (8) make the system reachable over the Internet by many users at once.

IV. SYSTEM DESIGN

A. Users and Roles Two roles are supported.

Two roles are supported. A driver is a registered user who offers seats in a vehicle; a passenger is a registered user who books them. The role is chosen at registration (Fig. 3b) and shown in the header of every page (Figs. 4–9). The same screens are reused with role-dependent content: for example, a driver's Booking Requests page lists incoming requests, whereas a passenger's My Bookings page lists outgoing ones.

B. Architecture

RideMate follows a conventional three-tier layout (Fig. 1). The browser, on a phone or computer, talks over HTTPS to a Flask application. Flask handles authentication, ride management, booking, notifications, tracking, SOS and rating, and stores everything in a PostgreSQL database. Map tiles, geocoding and routing are fetched from public services, and the device position comes from the browser itself. Table I lists the technologies used.

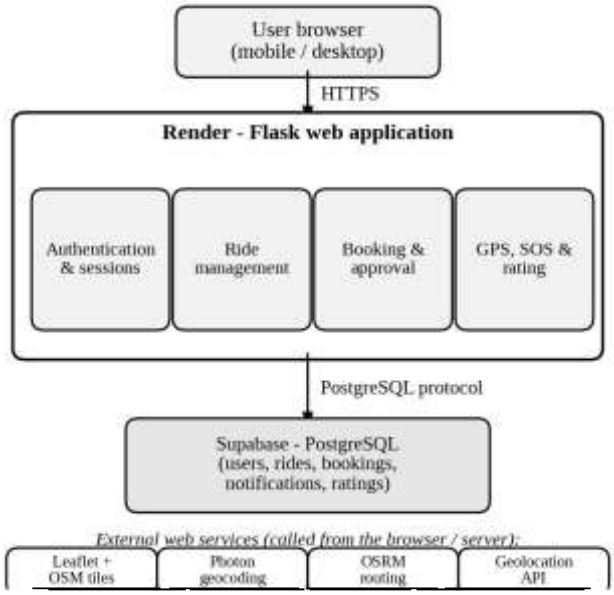


Fig. 1. Deployed architecture of RideMate: browser, Flask application on Render, PostgreSQL on Supabase, and external map services.

TABLE I-TECHNOLOGY STACK

Component	Technology
Language / back end	Python, Flask
Front end	HTML, CSS, JavaScript, Tailwind CSS, Font Awesome
Database	SQLite (prototype); PostgreSQL on Supabase (final)
Hosting	Render (web service)
Maps and routing	Leaflet, OpenStreetMap, OSRM, Photon geocoder
Positioning	Browser Geolocation API
Tooling	Git/GitHub, Visual Studio Code

C. Ride Lifecycle

Fig.2 summarises the workflow. After login a user reaches the dashboard and either offers or searches for a ride. A booking starts as Pending. The driver's decision moves it to Accepted or Rejected, and a notification is created for the passenger either way. Only accepted bookings can be tracked. When the destination is reached the ride is closed, the passenger is asked for a rating, and a charity option is shown.

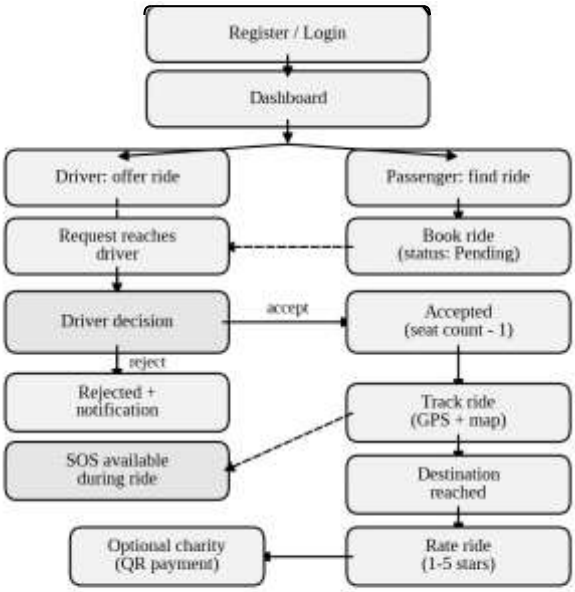


Fig. 2. Ride lifecycle and booking states. Dashed arrows show information passed between the two roles.

V. DATABASE DESIGN

The data model has five main tables (Table II). Users holds profile data, including the vehicle number and the emergency contact. Rides stores each offered ride together with its total and remaining seats. Bookings links a passenger to a ride and carries the status. Notifications

records events shown on the notification page, and Ratings stores one score per booking.

TABLE II - MAIN DATABASE TABLES

Table	Principal attributes
users	id, full name, student ID, email, department, year, role, password, vehicle number, emergency contact name and phone
rides	id, driver id and name, pickup, destination, date, time, total seats, available seats, vehicle number, status, created time
bookings	id, ride id, passenger id, status (pending / accepted / rejected), booking time
notifications	id, user id, title, message, read flag, created time
ratings	id, booking id (unique), ride id, passenger id, driver id, rating 1–5, created time

Foreign keys tie bookings to users and rides, and ratings to bookings. A rating is restricted to the range one to five, and the booking identifier is unique in the ratings table so that a passenger cannot rate the same booking twice. A driver's average rating can later be obtained by grouping ratings on the driver identifier.

VI. IMPLEMENTATION

A.Landing, Registration and Login

The public landing page *(Fig.*3a) explains the idea and links to Find Ride, Offer Ride and Login. Registration (Fig. 3b) asks for full name, student ID, college e-mail, department, year, role, and a password entered twice. Login (Fig. 4a) takes an e-mail and password; on success Flask starts a session and the user lands on the dashboard with a confirmation message. Protected routes check the session before serving a page.

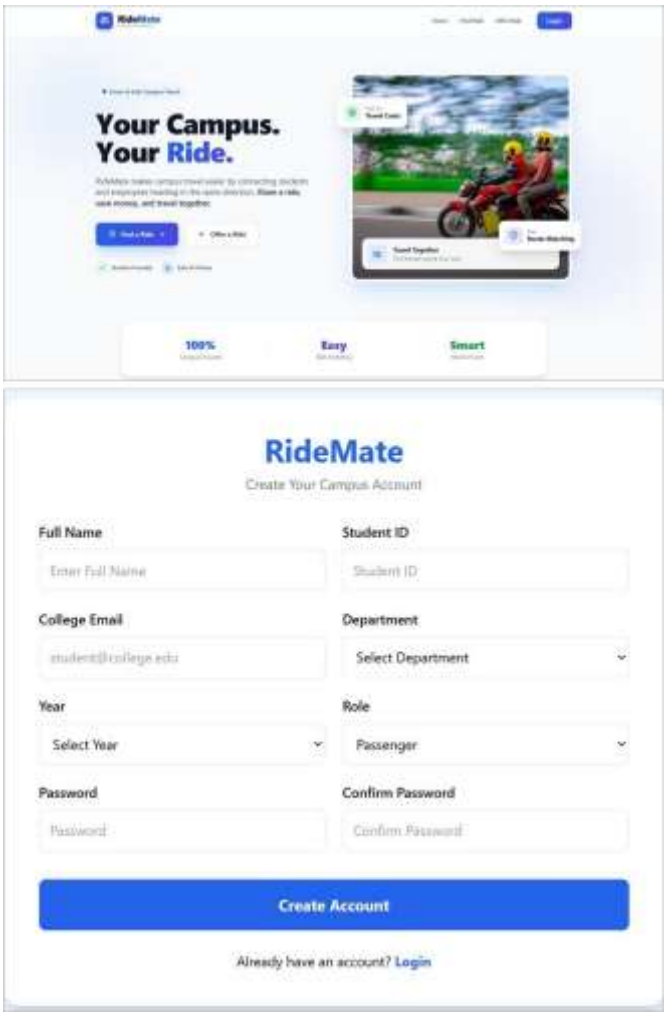


Fig. 3. (a) Landing page. (b) Registration form with department, year and role fields.

A. Dashboard

The dashboard (Fig.4b) is the hub. A sidebar gives access to Offer Ride, Find Ride, My Rides, My Bookings, Booking Requests, Notifications and Profile. Four counters summarise activity; in our test deployment they showed 12 active rides, 18 registered drivers, 31

registered passengers and 5 bookings. Two quick-action cards lead directly to offering or finding a ride.

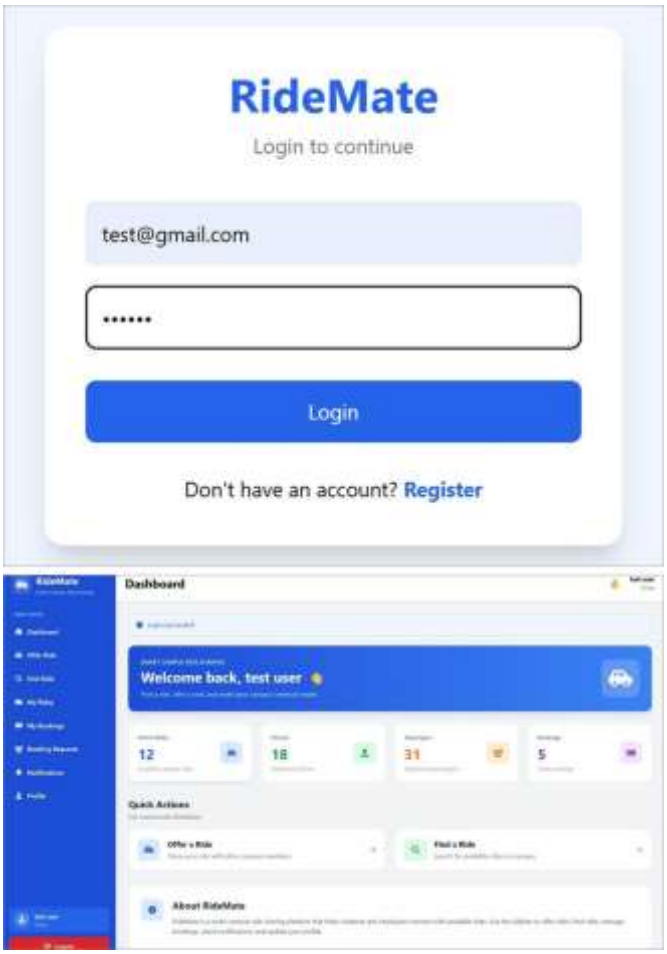


Fig. 4. (a) Login screen. (b) Driver dashboard with activity counters and quick actions.

A. Offering and Finding Rides

A driver fills in a short form (Fig.5a): pickup, destination, date, departure time, available seats and vehicle number. Submitting it creates a ride record with the seat count set to the offered value. Passengers see these rides as cards (Fig. 5b). Each card shows the driver, the from–to route, date, time, remaining seats and vehicle number, with a Book Ride button. A badge at the top gives the number of rides currently available.

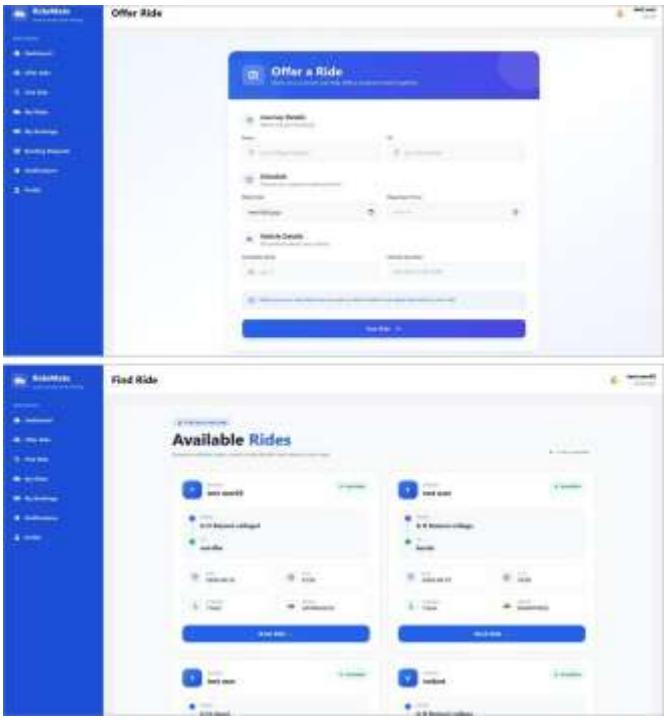


Fig. 5. (a) Offer Ride form for drivers. (b) Available Rides list as seen by a passenger.

B. Booking, Approval and Seat Accounting

Pressing Book Ride first validates the session and checks that the ride still has a free seat, then inserts a booking with status Pending. The driver sees the request under Booking Requests (Fig. 6b) and can accept or reject it. Accepting decrements the remaining seat count of the ride; rejecting leaves it unchanged. The driver's My Rides page (Fig. 6a) shows the effect: each ride carries a seat bar, and a ride whose only seat has been taken reads 0/1 with "1 seat booked". The passenger's My Bookings page (Fig. 7a) lists every request with its status, and offers Track Ride and Cancel Booking on accepted ones.

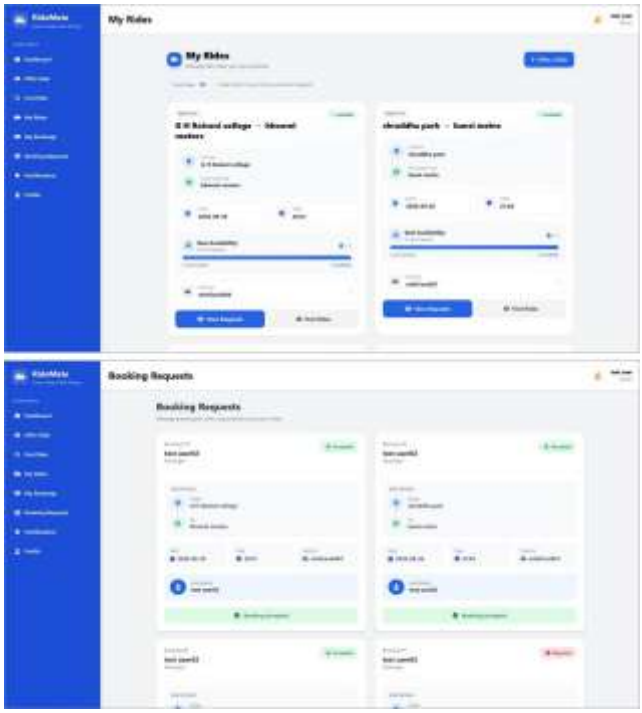


Fig. 6. Driver side. (a) My Rides with seat-availability bar. (b)Booking Requests with accepted and rejected entries.

C. Notifications

Each decision creates a notification for the passenger (Fig. 7b), for example “Ride Booking Accepted” or “Ride Booking Rejected”, with a read marker and a running count. This removes the need for a separate call or message to learn the outcome. Notifications are kept in the database, so they remain available after logout.

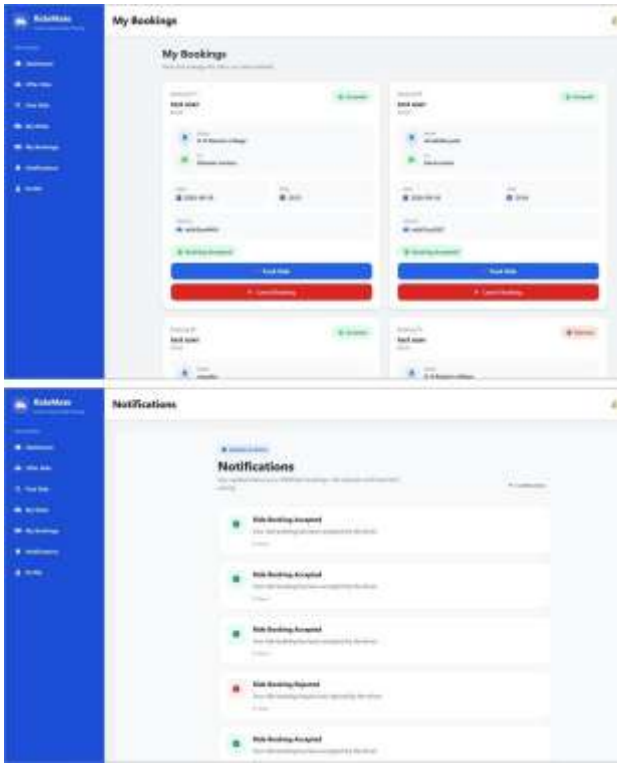


Fig. 7. Passenger side. (a) My Bookings with Track Ride and Cancel Booking actions. (b) Notification list.

D. GPS Tracking and Routing For an accepted booking, Track Ride opens the tracking page (Fig. 8a). It shows the pickup point, the route and the drop-off location, then a Leaflet

map on OpenStreetMap tiles. The destination text entered by the driver is geocoded with Photon, and the page reports whether the result lies inside the Nagpur region (“Destination found inside Nagpur”), which filters out obviously wrong matches for free-text place names. The passenger’s own position is requested through the browser Geolocation API after permission is granted, and the Find My Location button recentres the map. Distance and estimated time are computed from a route returned by OSRM and are shown beneath the map (Fig. 8b); the page also repeats the driver, vehicle, date, time and booking status. While tracking is active, the distance between the latest position fix and the destination is compared with a small threshold. When it falls below the threshold the ride is marked complete and the completion page is opened. A tolerance is needed because consumer GPS readings fluctuate by several metres and indoor or urban-canyon conditions can make them worse.

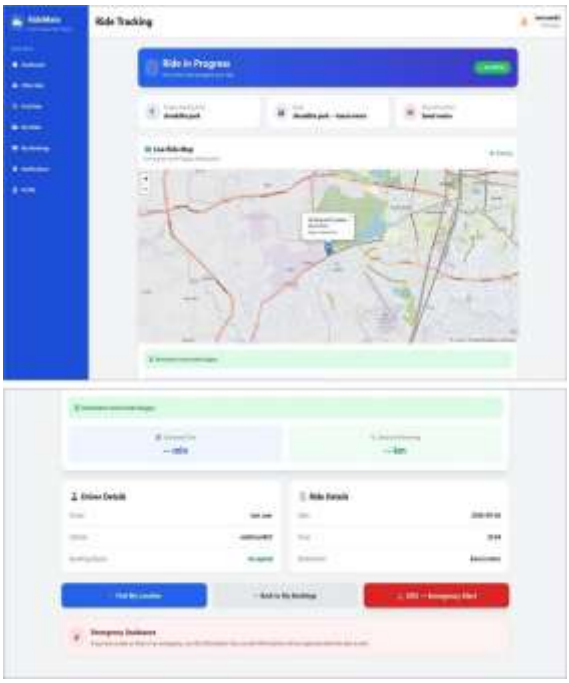
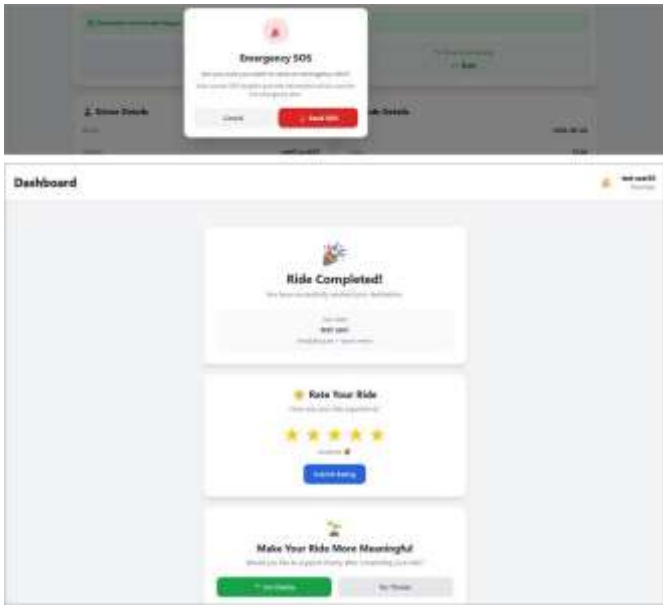


Fig. 8. Ride tracking. (a) Live map with pickup, route and drop-off. (b) Distance and time panel, driver and ride details, and the SOS control.

E. Emergency SOS

A red SOS button is always visible on the tracking page. Pressing it opens a confirmation dialog (Fig. 9a) that states the current GPS position and ride information will be used for the alert; the two-step action reduces accidental triggers. The emergency contact that the alert is associated with is stored in the user profile (Fig. 10). The profile page also states plainly that this contact is used only when SOS is triggered during an active ride, which we considered important for user acceptance.

F. Completion, Rating and Charity On arrival the passenger sees a Ride Completed page with the route and driver (Fig. 9b). A five-star control with a text label (“Excellent!” at five stars) lets the passenger submit a rating, which is saved against the booking, ride and driver. Afterwards the page offers Do Charity or No Thanks. The first opens a separate page that displays a payment QR code; the payment itself is made in the user’s own payment application and is outside RideMate.



G.

Fig. 9. (a) SOS confirmation dialog. (b) Ride Completed page with five-star rating and optional charity.

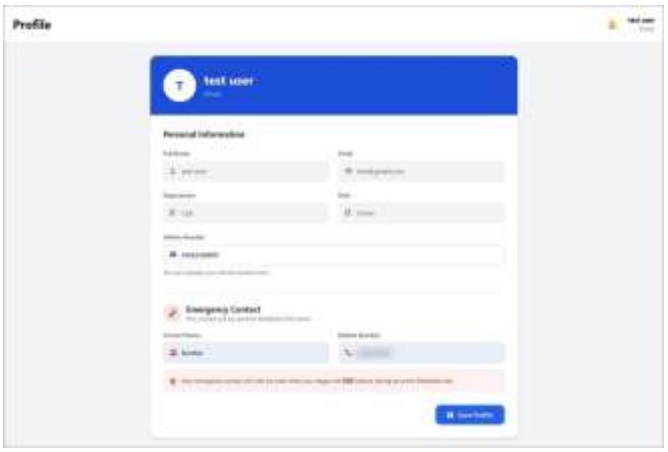


Fig. 10. Profile page with vehicle number and emergency contact (phone number blurred).

VII. DATABASE MIGRATION AND DEPLOYMENT

The first version used SQLite. It needs no server, stores data in a single file and is well suited to quick development on one machine. It is less suited to a public service where many users write at the same time, and a file on a developer's laptop cannot be shared by a hosted application. For the final system the schema was therefore moved to PostgreSQL, a client–server database with foreign keys, constraints and good support for concurrent connections [11]. Supabase provides the managed PostgreSQL instance [12], and the Flask application runs as a web service on Render [13], so that all users, wherever they connect from, read and write one central database.

The change affects more than the connection string. SQL placeholders, auto-increment columns, date and time handling and error types differ between the two systems, and each query that worked on SQLite had to be checked after the switch. Keeping the database access in a small number of functions made this manageable.

VIII. TESTING AND RESULTS

Functional testing was done by hand with two accounts, a driver ("test user") and a passenger ("test user02"), exercising each module from registration to rating. Table III lists the cases and the behaviour expected from the system.

TABLE III-FUNCTIONAL TEST CASES

Table	Expected result
Valid registration	User account created
Duplicate e-mail	Registration rejected
Valid / invalid login	Dashboard opens / error shown
Offer ride	Ride record created and listed
Find ride	Available rides displayed
Book ride	Booking created as Pending
Driver accepts	Status Accepted; seat count reduced
Driver rejects	Status Rejected; seats unchanged
Notifications	Matching notification shown to passenger
Track ride	Map and ride details displayed
SOS	Confirmation dialog, then alert action
Destination reached	Ride Completed page opens
Rating / duplicate rating	First stored (1–5); second rejected
Do Charity / No Thanks	QR page opens / normal flow continues

The screenshots in Section VI come from these runs and show the intended behaviour end to end. The driver account had posted ten rides (Fig. 6a). Bookings numbered 7 to 11 for the passenger account include both accepted and rejected requests (Figs. 6b, 7a), and the passenger's notification page lists five entries that match them (Fig. 7b). A ride with one seat shows 0/1 once its booking is accepted, confirming that seat accounting follows the approval step. The tracking, SOS, completion and rating pages were reached through the accepted booking for the ride from shraddha park to bansi metro (Figs. 8, 9).

No measurements of response time, concurrency or positioning accuracy were collected, and we make no claim about emissions or cost savings; those would need field data from real commuters.

IX. DISCUSSION AND LIMITATIONS

The system covers the whole ride from offer to rating, and the driver-approval step keeps control with the person who owns the vehicle. The hosted setup allows several people to use the application at once, which a local database could not.

Several limitations remain. GPS accuracy depends on the device and surroundings. Geocoding and routing depend on public services whose availability and rate limits are outside our control. Place names are typed as free text, so two drivers may spell the same stop differently, which weakens search. Identity is tied only to an e-mail address and student ID entered at sign- up; no verification step confirms them, and vehicle numbers are not checked against any registry. The SOS flow records and associates the alert with a contact, but it does not yet push a message to that contact. Fraud detection is absent, and the platform has not been load-tested for a large user base.

X. FUTURE WORK

The most useful next steps are, in our view: verified college e-mail and driver documents; a real-time SOS notification by SMS or push message; live driver location rather than passenger-only tracking; route-aware matching that checks whether a passenger's pickup lies along a driver's path; an administrator dashboard for reports and incidents; standardised stop names or map-based pickers; and analytics such as shared trips, average occupancy and peak hours. A pilot with real students would allow the effect on individual trips and travel cost to be measured properly.

XI. CONCLUSION

RideMate is a campus ride-sharing application that joins ride offering, searching, booking with driver approval, notifications, GPS tracking, SOS, rating and a charity option in one web system. Moving from a SQLite prototype to PostgreSQL on Supabase, with the Flask service on Render, turned a single- machine demonstration into something several users can reach through a URL. The work is a working design and implementation rather than a measured study; its value lies in showing how these pieces fit together for a campus setting and in providing a base for the extensions listed above.

ACKNOWLEDGMENT

The authors thank [Asst.Prof. Surabhi Chavhan] and [G H Raisoni University] for guidance and facilities during this project.

REFERENCES

[1] M. Furuhata, M. Dessouky, F. Ordóñez, M.-E. Brunet, X. Wang, and S. Koenig, "Ridesharing: The state-of-the-art and future directions," Transp. Res. B, Methodol., vol. 57, pp. 28–46, Nov. 2013.

[2] N. Agatz, A. Erera, M. Savelsbergh, and X. Wang, "Optimization for dynamic ride-sharing: A review," Eur. J. Oper. Res., vol. 223, no. 2, pp. 295–303, Dec. 2012.

[3] N. D. Chan and S. A. Shaheen, "Ridesharing in North America: Past, present, and future," Transp. Rev., vol. 32, no. 1, pp. 93–112, 2012.

[4] S. Shaheen and A. Cohen, "Shared ride services in North America: Definitions, impacts, and the future of pooling," Transp. Rev., vol. 39, no. 4, pp. 427–442, 2019.

[5] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," Proc. Natl. Acad. Sci. U.S.A., vol. 114, no. 3, pp. 462–467, 2017.

[6] M. Haklay and P. Weber, "OpenStreetMap: User-generated street maps," IEEE Pervasive Comput., vol. 7, no. 4, pp. 12–18, Oct.–Dec. 2008.

[7] V. Agafonkin, "Leaflet: An open-source JavaScript library for interactive maps." [Online]. Available: https://leafletjs.com

[8] D. Luxen and C. Vetter, "Real-time routing with OpenStreetMap data," in Proc. 19th ACM SIGSPATIAL Int. Conf. Adv. Geogr. Inf. Syst., Chicago, IL, USA, 2011, pp. 513–516.

[9] Komoot, "Photon: Open-source geocoder for OpenStreetMap data." [Online]. Available: https://photon.komoot.io

[10] A. Popescu, "Geolocation API specification," W3C Recommendation. [Online]. Available: https://www.w3.org/TR/geolocation/

[11] PostgreSQL Global Development Group, "PostgreSQL documentation." [Online]. Available: https://www.postgresql.org/docs/

[12] Supabase Inc., "Supabase documentation: Database." [Online]. Available: https://supabase.com/docs

[13] Render Services, Inc., "Render documentation: Web services." [Online]. Available: https://render.com/docs

[14] M. Grinberg, Flask Web Development: Developing Web Applications with Python, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2018.