ISNNI: 3007-3391 HEC Category: Y

Journal of Computers and Intelligent Systems
Vol.4, No.1, 2026, Pages 45 - 49
Journal Homepage:
https://journals.iub.edu.pk/index.php/JCIS/

Smart Hitchhiking: A Real-Time Dynamic Solution to optimize urban commutes

Haseeb Khalid[1], Syed Hassan Ali[1] , Muzamil Munir[1], Takreem Raza Shah[1], Shazia Usmani[1], Uzma Afzal[1]

[1] *Federal Urdu University of Arts, Science and Technology, Karachi, Pakistan*

ARTICLE INFO

Article History:

Received:	October	11, 2025
Revised:	January	01, 2026
Accepted:	March	20, 2026
Available Online:	March	30, 2026

Keywords:

Ride-sharing
Urban metropolis
Hitchhiking
Real-time
Commute optimization

Classification Codes:

Funding:

This research received no specific grant from any funding agency in the public or not-for-profit sector.

ABSTRACT

Escalating population growth of the urban metropolis introduces many issues and challenges such as traffic congestions and rising travel costs along with environmental pollution. To reduce and fix the impact of these issues ride-sharing solution has emerged as a potential candidate; existing solutions often fall short due to static ride-matching models, delays in communication, and no real-time adaptability. This paper presents **Smart Hitchhiking (SHi)**—a dynamic and real-time ride-sharing solution designed to cater the limitations of existing market by optimizing urban commutes. The proposed solution has state-of-the-art features such as overlapping route identification, partial trip matching, and route correlation analysis, moving beyond traditional origin-destination models. Developed as a modular client-server architecture with React and MySQL, SHi improves scalability, real-time interaction, and secure data handling. The system is evaluated on an AVD emulator which ensures that SHi offers an efficient, adaptive, and user-friendly alternative to existing ride-sharing applications.

Corresponding Author's Email: *uzma.afzal@fuuast.edu.pk

1. Introduction

The rapid increase in population and urban density of Karachi has led a huge rise in of private cars on roads. This massive vehicle influx creates a severe traffic issue, affect fuel usage, raises travel cost high and disturbs city environment. Ride-sharing has emerged an alternate solution to cater these traffic challenges and reduce the travel cost. It also helps ease the traffic congestions on crowded roads. Ride-sharing approach also reduces vehicle occupancy rates to ease the burden on public transportation system. In the past decade advanced mobility methods such as demand based ride-sharing and Mobility as a Service (MaaS) have gained popularity for daily traveling. Many ride-sharing solutions like Uber [1] and Lyft [2] help to reduce traffic congestions and environmental pressure but can lead to unintended consequences [3]. Research studies also explore the negative impacts. These ride-sharing solutions can increase the urban traffic by replacing trips that might involve public transportation or walking. Moreover, deadheading (driving without passengers) can add emissions traffic congestion and wastage of fuel and resources. To cater these challenges researchers suggest a combination of private ride-sharing with public transit to promote pooled-rides and reduce traffic congestion [4].

In this study, the term "ride-sharing" specifically refers to a shared use of a motor vehicle by persons traveling along the same route. This model allows passengers to share the cost without making a profit for the driver by utilizing the advanced technology for optimal matching of routes, cost efficiency, and safety measures.

Ride sharing also offers an alternative aside from transport that simultaneously helps in reducing the environmental impact and transportation cost. Despite the numerous benefits that ride-sharing offers, prior research suggests several barriers that prevents its widespread adoption including economic, business, and technological constraints.

In large cities with high trip density, ride-sharing can significantly ease congestion, especially during peak hours. However, in smaller cities, limited sharing may lead to longer travel distances, potentially offsetting these benefits. Mostly ride sharing systems use ride-matching algorithms and dynamic pricing models to get the social and environmental benefits but still many studies highlight the limitation of influencing user adoption and the barriers to implementing these services [5]. Existing research studies explore the weaknesses in the current available solutions. An extensive analysis is done to find the actual research gap. Static ride sharing solutions do not facilitate users in real-time, they just provide user generated ride offers. Updates like change in driver schedule/ route do not propagate promptly which lead communication delay. To overcome the limitations of static ride sharing applications many dynamic solutions are explored and employed to facilitate the users. They use modern communication technologies for rapid interaction between stakeholders (detailed study is presented in the next section).

An extensive analysis of both static and dynamic solutions develops a foundation of our proposed solution with a rich set of features and functions. Smart Hitchhiking (SHi) is an attempt to fix the issues of existing local market such as match rate, scalability, distance covered, response time and matching latency. Traditional ride sharing solutions need to improve their route matching mechanism by minimizing the manual interventions. Dynamic ride sharing solutions achieve high match rate by employing optimization algorithms but their real world deployment is very limited. Similarly, in tradition ride sharing driver can deviate manually which effects the detour distance. Due to the user browsing and computational complexity, both ride sharing solutions have high response time. Matching latency of traditional ride sharing is low because it is based on users driven decisions. Similarly the simulation based modeling of dynamic sharing poses serious concern on system scalability. This technical analysis lays the foundation for SHi; a modern and dynamic ride-sharing app that addresses these gaps by incorporating the following distinguish features:

- Overlapping routes identification by utilizing common path between the user and driver.
- Instead of using same origin/destination to match routes calculates a correlation between paths.
- Match partial trips.

Section 5.1 provides a detailed comparison of SHi and other market solutions to justify the proposed solution. For the implementation of SHi, a client-server model is designed using React and MySql. All sensitive information is stored on a remote server. A modular approach with a control module and a separate GUI is developed to maintain and enhance system easily. To test and evaluate the proposed SHi AVD manager emulator is used.

The rest of the paper is structured as follows: Section 2 presents the background and related work. Proposed solution is discussed in Section 3. The implementation perspectives are presented in Section 4. A detailed discussion on proposed solution and traditional applications is presented in Section 5, privacy concerns along with the SHi limitation are discussed in Section 6, and the paper is concluded in the final section.

2. Background and Related Work

Hitchhiking is a mode of transportation in which user asks individuals to share ride. Two types of hitchhiking apps are available, i.e., static and dynamic. Several researches are conducted to explore the potential solutions for hitchhiking and ride-sharing applications. Identification of the real-world research gap is very important in order to recognize the actual issues and challenges of end users. It also provides a solid building block to design the architecture of our proposed solution. For this, a detailed research study is conducted.

In [6], authors discuss the development of micro-mobility and shared mobility platforms. An extensive discussion is presented for a comprehensive study of shared mobility innovations, such as car sharing and ridesharing. Optimization techniques for dynamic ride sharing are discussed in [7], with the real-time matching ride-sharing analysis. The paper provides insights into the complexities of dynamic routing and matching in real-time applications.

Static hitchhiking applications typically rely on user-generated ride offers and requests that do not update in real-time. These platforms serve as bulletin boards where users can post their travel plans, but they lack immediate responsiveness to changing conditions. For instance if a driver alters a schedule but does not update

the post can lead to an outdated information for potential passengers. Several static hitchhiking applications have emerged worldwide with the following distinguished features:

- User Interaction: Users can post their rides and also respond to the requests from other passengers by building a rider community.
- Data Presentation: These applications display a large volume of data. Multiple search filters are available to limit the specific preferences. Although users have to wait for an extended period of time due to the static nature.

[8] presents a carpool and hitchhiking website interface to organize rides into different categories, i.e., new rides and last-minute rides. It also creates a social network for hitchhikers. Digihitch.com is a Europe-based hitchhiking interface. It requires passengers to contact drivers for ride details that sometimes resulting in a complex user experience [9]. Easy Lift is a Pakistan-based carpooling platform. It allows potential users to post rides and explores similar routes. It lacks real-time updates but provides an affordable and environmental friendly solution to a traditional travel. Easy Lift includes basic features like route matching and profile verification along with and building a reliable ride-sharing community [10].

Humsafar RideShare (HRS) is also a local app which supports pre-scheduled ride-sharing by connecting drivers and passengers. It is designed for a cost-effective commuting similar to a bulletin board. Potential users can post their travel and ride plans in advance. HRS ensures user security by incorporating the verification of profiles and utilizing a secure connection [11]. Besides, all the listed benefits static ride sharing still has certain limitations:

- Static systems suffer from real time updates. They can not accommodate changes/updates in users plan. Drivers may also forget to change/update the offers to make apps unreliable [12].
- Static systems lack rapid communication mechanisms. Interaction among stakeholders occurs through email and messages which delays the response times specifically for last-minute requests [13].

Dynamic hitchhiking applications are equipped with real-time interactions which speed up the communication among different stake holders. Applications are available 24/7 so users can easily update/change their initial travel plans to find matches on the go. Primary key features of dynamic hitchhiking applications are:

- A real-Time Interaction among users to communicate/adjust plans easily which makes dynamics apps more flexible.
- They utilize GPS technology to fetch the precise location finding/tracking. Based on the current traffic conditions they offer optimized routes to traveler.

Bykea offers on-demand motorbike rides and parcel delivery, allowing users to book rides dynamically with real-time location tracking [14]. Careem's carpooling feature supports dynamic ride-sharing, enabling users to share rides and split costs, adding flexibility to urban commutes [15]. Uber operates a dynamic ride-sharing model where users can book rides instantly, and its "UberPOOL" feature allows riders heading in the same direction to share trips and costs [1]. This in-depth study of existing ride sharing systems provides us the base to architect our solution. We aim to design a framework with a rich set of features without compromising on its technical soundness, affordance and user experience. In section 5, we present an evaluation of our proposed solution to show how it caters the limitations and issues of the existing solutions.

3. Smart Hitchhiking (SHi): The proposed Solution

Smart Hitchhiking (SHi) is a smart solution by fixing the issues and limitations of current market solutions analyzed in the previous section. SHi is a unique dynamic ride-sharing app:

- It identifies overlapping routes by taking into consideration the common path between the ride provider and the ride seeker.
- It does not rely on having the same origin or destination to match routes but instead identifies a correlation between the paths, even if they start and end at different locations.
- It allows the system to match trips for users, even if they only share part of the route.

SHi work flow initiates after the installation of the given binaries. The very first step from the ride seeker/provider perspective is registration. A valid registration acquires personal data and vehicle data (provider). It also includes multiple identifiers that the program will be referring to when merging routes. Vehicle

data which is entered by the user is saved and appeared as the default information for the user's vehicle when creating a route.

Once the user profile is created, two views are available, i.e., driver and passenger. The driver view is primarily used to set a route, while the passenger view is used to join someone.

3.1 Driver and Passenger Views

The driver view allows users to get registered as drivers by uploading all the documents of the vehicle and the driver. A valid driver's license is compulsory for registration. Under Driver Mode, the driver can establish routes with key information that includes to and from locations. The "to" location defaults to the driver's current location. Drivers can plan and save routes for later use; it becomes simple to define the common traveling paths as well as preferred schedules. Once all information is entered, the route details are saved and appeared to passengers.

By using a passenger view, a ride can be requested. Passenger need to share his source location (the default value of location acquired from GPS), destination location and preferred departure time. After entering the required information, system starts its work by applying algorithms with the objective of identifying the best possible matching routes. The passenger only view relevant rides after choosing a route and a meeting point from where he will join the ride.

3.2 System Architecture Design of SHi

A client-server model is developed to ensure that all crucial and sensitive information is securely stored on a remote server rather than the mobile device. The device retains only user preferences, preserving storage space and enhancing security. A modular approach is adopted for flexibility and software maintenance. Application components are organized as per the specific functions. Primary components of SHi are:

- **Control Modules:** A matching Algorithm and a strong web service access are the main uniqueness of SHi. Control module is developed as a core component to implement the ride-matching algorithm which enables users to connect with drivers or passengers on similar routes. It also manages the access to external services such as Google Maps (API) and supports the route mapping and location-based functionalities.
- **Data Input/Output Module:** This module manages data interfacing/exchange between the application and the central database. It processes incoming and outgoing data requests, handling storage and retrieval functions to ensure that user and route information is updated efficiently.
- **Graphical User Interface (GUI):** The GUI is designed to separate the application's logic and user interface. It also ensures a smooth and intuitive user experience. The display component handles all user interactions and visual elements, ensuring that the application remains user-friendly and accessible.

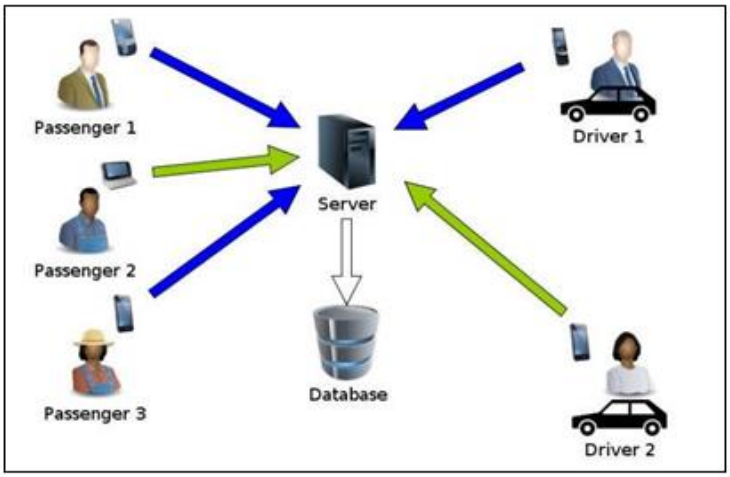

Figure 1. Matching Routes for the Users (Same color Arrows)

The relationship between these components is illustrated in the Figure 1, which shows how the central control modules interact with the database to find matching routes, the GUI for user interaction, and other mobile functionalities such as GPS for location tracking. The primary functionalities of the control module are:

- Registration and Configuration: Manages user sign-up, login, and personal settings.

- Adding and Searching Rides: Allows users to create new ride entries and search for rides based on specific origins and destinations.
- Route Matching and Binding: Processes user requests for shared routes, finding the most efficient match between drivers and passengers.
- Trip Monitoring: Tracks active trips in real time, enhancing user safety and providing updates.

The system management is responsible for initiating all core functions, some of which are user-driven (like adding or searching for rides), while others, such as route matching and trip monitoring operate automatically to streamline the user experience. This modular and systematic design ensures that each component works cohesively, providing a reliable and efficient ride-sharing service (Figure 2).

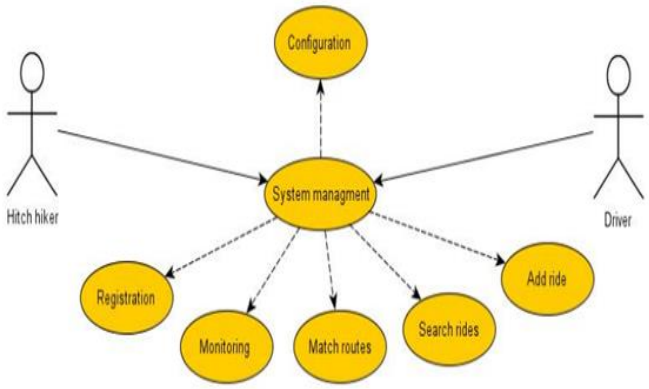

Figure 2. The Core Functionalities of the Proposed Solution (SHi)

4. Smart Hitchhiking (SHi): The Implementation Perspective

SHi is built using React Native language [16] for Android based devices. AVD manager emulator [17] is used for the testing purpose. The selection of the Android is also based on certain reasons. Android is supported by various types of devices with 71.65% of the global smartphone market share as of November 2024[18]. Moreover, its open source approach grant an excess to the phone's internal functionalities. Figure 4 shows the implementation flow of SHi.

4.1 Algorithms and Database

This ride-matching algorithm is designed for a ride-sharing app and integrates with an SQL database (e.g., MySQL) to match drivers and passengers based on similar starting and ending points within a predefined deviation range. The algorithm retrieves the driver's and passenger's source and destination coordinates, then calculates stopping points along each route. It leverages SQL queries to search the database for routes with proximity within the driver's allowed deviation from the source and destination. When a potential match is found, the passenger's pickup and drop-off points are appended to the driver's route, creating a combined path. The optimized routes are stored in the database, ensuring efficient access for multiple users in real-time. The algorithm prioritizes the longest matching routes that fall within the deviation limit, helping drivers find optimal matches while minimizing detours.

To formulate the ride matching algorithm, Haversine distance is used (Algorithm 1). It calculates the circle distance between two points using their latitude and longitude coordinates. Using 6,371 km as the standard Earth radius, latitude / longitude differences are converted from degrees to radians. $a calculates the square of half the chord length between the points. $c calculates the angular distance in radians. The final multiplication by Earth's radius gives the distance in km. For instance:

```
$distance = haversine_Distance(40.7128, -74.0060, 34.0522, -118.2437);
Returns ~3,935 km (distance between NYC and LA)
```

The match score is calculated based on distance and time difference. Distance constitutes 50% weight, while time difference contributes 30%. The closer the distance and time, the higher the score. The maximum possible score is 80%.

Algorithm 01: A Mathematical Formulation of Ride Matching

Haversine distance Calculation between two points.

```
private function haversineDistance($lat1, $lon1, $lat2, $lon2)
{
    $earthRadius = 6371; // Earth's radius in km
    $dLat = deg2rad($lat2 - $lat1);
    $dLon = deg2rad($lon2 - $lon1);

    $a = sin($dLat / 2) * sin($dLat / 2) +
         cos(deg2rad($lat1)) * cos(deg2rad($lat2)) *
         sin($dLon / 2) * sin($dLon / 2);

    $c = 2 * atan2(sqrt($a), sqrt(1 - $a));
    return $earthRadius * $c; // Distance in km
}
```

Calculate the match score based on distance and time difference.

```
private function calculateMatchScore($distance, $timeDiff,
                $maxDistance, $maxTimeDifference)
{
    $distanceScore = max(0, (1 - ($distance / $maxDistance)) * 50);
    // 50% weight
    $timeScore = max(0, (1 - ($timeDiff / $maxTimeDifference)) * 30);
    // 30% weight
    // Add more criteria for scoring if needed
    return $distanceScore + $timeScore; // Total score in percentage
}
```

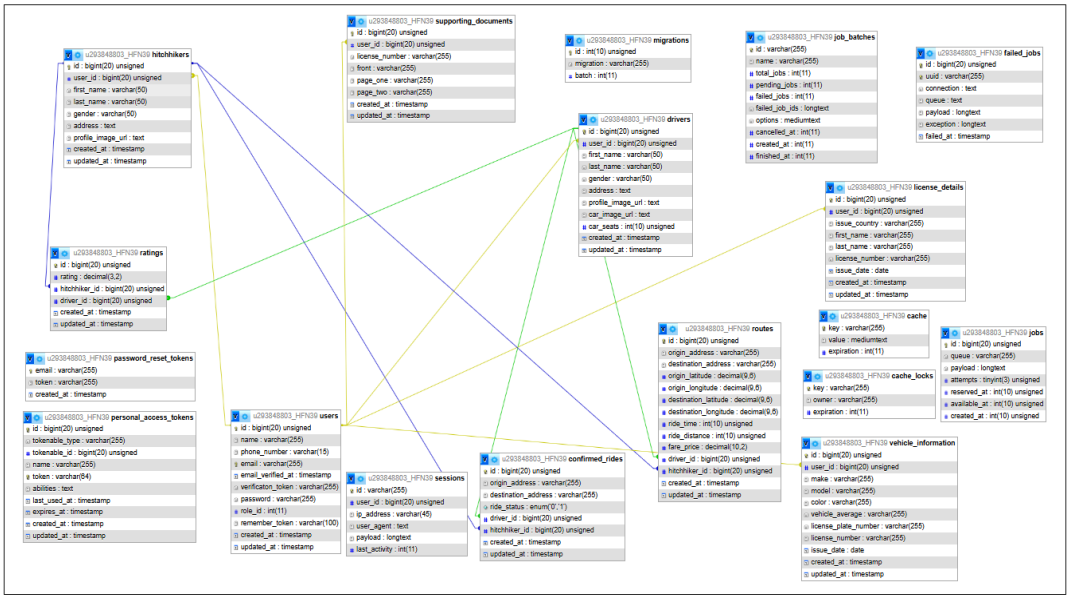

Figure 3. SHi- A DB Schema

The database for the ride-matching algorithm is built to store multiple users along with their routes, but there are also other additional tables to capture user preferences and groups (Fig. 3). The most important table is the Routes table, which contains all the routes parsed from the Google Maps API in the format of coordinates. We create routes as unique IDs and store them in multiple steps, denoted location points, which are represented by latitude and longitude coordinates. This format enables us to compute distances between the points of interest through geometric formulas, thus matching close but not identical points with the possibility of flexible matching within a specified radius.

- Routes Table: Each route entry includes a unique route ID, coordinates, and optionally a name (e.g., intersection or landmark). It references the driver's details from the Users table. As passengers join, their details are referenced in a Route-Users Relation table to track which passengers are linked to each route.

- Users Table: This table stores user details, including status, ratings, and reviews. Each user can belong to multiple Groups, which helps to create matches within specific groups if necessary (e.g., university students or corporate employees). The user's group memberships are stored in the Group-User Relation table.
- Route Preferences and General Preferences Tables: These tables store users' specific preferences for a given route (e.g., non-smoking, music preference) and their general preferences across all rides, respectively, to improve the matching experience.

We are not discussing remaining tables here to avoid too much details, moreover Fig 3 is self-explanatory and easy to understand for a DB developer. This database is implemented in MySQL for external storage, allowing for scalability and efficient query handling. SQLite is used on the device for quick, local data storage, offering offline functionality for user preferences and local caching of routes. This structure facilitates efficient route and user matching while also providing flexibility for grouping, user preferences, and real-time updates in ride-sharing scenarios. The database stores user locations and timestamps and Google Maps APIs are further used for maps and routing. GPS updates are used for the real-time tracking.

Step 1: Install Required Packages

```
npm install @react-native-community/
geolocation axios
```

Step 2: Geolocation Utility

```
import Geolocation from '@react-native-community/
geolocation';
export const getCurrentLocation = () =>
 new Promise((resolve, reject) =>
 {  Geolocation.getCurrentPosition(
    position => resolve(position.coords),
    error => reject(error),
      { enableHighAccuracy: true, timeout: 20000,
        maximumAge: 1000 });
 });
```

Step 3: API Service

```
import axios from 'axios';
const API_URL = 'https://your-backend-api.com';
export const findMatchingRoutes = async (driverSource,
driverDestination, maxDeviation) => {const response =
await axios.post(`${API_URL}/match-routes`,
{driverSource,  driverDestination, maxDeviation});
 return response.data; };
```

Step 4: Ride Matching Component

```
import React, { useState } from 'react';
import { View, Text, Button, TextInput, Alert }
from 'react-native';
import { getCurrentLocation } from './locationService';
import { findMatchingRoutes } from './apiService';
const RideMatchingScreen = () =>
{const [driverDestination, setDriverDestination] =
 useState('');
 const [matches, setMatches] = useState([]);
 const handleFindRoutes = async () => {
  try {const driverSource = await getCurrentLocation();
   if (!driverDestination) return Alert.alert('Destination
       required');
   const matchedRoutes = await findMatchingRoutes
   (driverSource, driverDestination, 2);
   setMatches(matchedRoutes);}
   catch {Alert.alert('Error', 'Could not find matching
routes.');}}; return (
 <View style={{ padding: 20 }}>
 <Text>Ride Matching</Text>
 <TextInput placeholder="Enter destination"
onChangeText= {setDriverDestination} />
 <Button title="Find Matching Rides"
onPress={handleFindRoutes} />{matches.map((match, i)
=> ( <Text key={i}>Route {i + 1}: {match.details}
</Text> ))}   </View>);};
   export default RideMatchingScreen;
```

Figure 4. An implementation work flow of Smart Hitchhiking

This component allows users to input a destination, finds nearby matches based on current location, and displays available routes. Adjust backend logic for route calculations.

4.2 User Interaction: A Scenario

Figure 5 shows the user interaction. System operates in, by making a match with people who would likely share a ride together among users and the system. Once the driver inputs his origin and destination, the system saves the route chosen and waits for a passenger to initialize a search for a ride according to the desired locations. Once a suitable route is found for the given passenger's path, the comparison of passenger and drivers' route is done for the optimal match, according to destination, departure time, and estimated arrival time. After that, another algorithm is activated to find similar paths and subpaths for comparison. If the system finds a similar match; it links those riders and starts ride tracking. System continues tracking till the completion of ride.

System bounds driver to accept his last route, because there are numerous possible routes between the same origin and destination, and he might want to follow one route rather than another, unlike the passenger who does not usually have a preferred route. This scenario caters the involvement of one user and one driver, but the same process repeats with the multiple users too. The relationships between multiple drivers and passengers is stored in a centralized database. Each driver is represented by a uniquely colored arrow, symbolizing a distinct entry in the routes table. Passengers can connect to one of these drivers, adopting the same unique symbol (in this case, the same arrow color) as their designated driver.

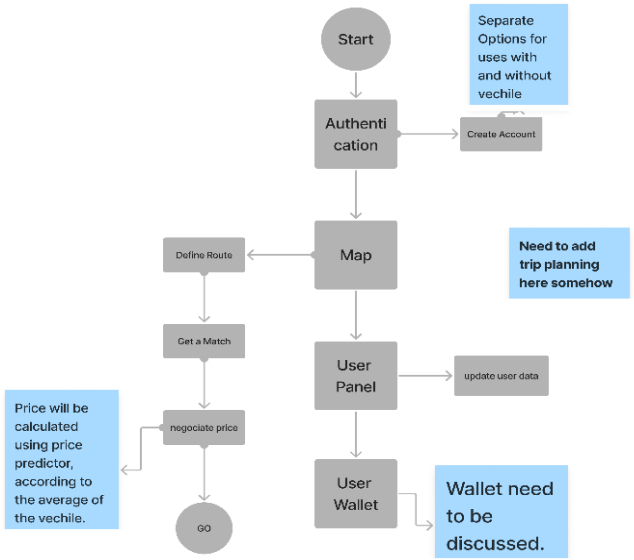

Figure 5. System and User Interactions

5. SHi: Performance Evaluation

In this section the advantages of using SHi are discussed from technical and user point of view. To evaluate the SHi, a controlled testing environment was setup at the university premises to replicate the real world scenario. Students, faculty members and staff participated in this evaluation. User feedback was acquired using a web interface. Moreover, before deploying SHi in this testing setup. A group of developers generated the different internal test cases to ensure the correctness of the core logic of system. They verified that user rides are only matched when both conditions (a 3 km pickup radius and sufficiently similar route) were satisfied. They also tested SHi for Integration to ensure the seamless working of all components such as user interfaces, backend logics with associated database. To check the system response multiple ride requests were generated, efficiency of handling repeated operations was evaluated by analyzing the system response time. After all these verification

checks, system was deployed to the above-mentioned testing environment for evaluation the interaction with actual real world user to gauge the SHi usability. Each user entry recorded the pickup and drop-off coordinates, route dynamics and estimated travel time. User locations were randomly generated using a defined urban area, clustering was used to mimic real-world scenarios such as office areas, residential zones, and peak-hour demand hotspots. Travel times were calculated using distance-based estimations, and routes were approximated using shortest-path logic. These control knobs ensured that data was not completely random, making SHi evaluation meaningful. The results of user feedback was also validated the usability, performance and efficiency of SHi. 83.3% users validated the matching efficiency by showing how effectively the system pairs users. The quick response of the system to cater the matching requests was measured as latency. The fastest, slowest and average response time was 120,180 and 130 ms respectively. User experience with SHi was Each user feedback was recorded on a scale of 1 to 5 based on factors such as ease of use, response time, and quality of matches. User rating in the testing setup was between 3.5 and 4.5. The feedback validated the system responsiveness and ride matching relevant to the testers need.

5.1 A Comparative study

Optimized Route Matching: The system utilizes a specially developed algorithm to identify an optimal route shared between a driver and multiple passengers. Unlike traditional car-sharing applications where passengers manually select from a list of drivers, our application automatically calculates the best overlapping route. Routes are prioritized based on criteria such as the nearest common path, proximity to origin and destination, and other factors, ensuring that passengers are offered with the most relevant options.

User-Friendly Interface (GUI): The application's GUI is designed to be intuitive, minimizing disruption to users' regular mobile activities. SHi focused on simplifying interactions to reduce the number of steps required to book a ride. This streamlined design improves usability, enabling users to accomplish their tasks efficiently.

Real-Time Location Tracking: Both driver and passenger locations are displayed in real time on a map, enhancing reliability and user experience. By allowing users to see each other's locations, this feature contributes to improved safety and facilitates easier recognition at pickup points

Enhanced Personal Security: Recognizing the openness of the platform and the potential anonymity of users, the system integrates features that prioritize both user-friendliness and security. One such feature is a unique registration code required to activate the application. This code can be obtained via several methods, the most common being through invitation by a friend who is already a registered member with enough credits to generate codes. Other approved groups, such as student associations or company networks, may also distribute registration codes, helping to create a trusted user base.

User-Contributed Drop-Off Recommendations: The system displays recommended drop-off locations, which can be modified or updated by users. This collaborative feature keeps the platform dynamic and adaptable to real-time changes, offering improved convenience for passengers and drivers alike.

Drivers motivation: The motivation for drivers to offer car rides can be achieving rating points (grant discounts and offers). Payment from passengers is another option (although not entirely allowed everywhere, because of restrictions of insurance policies). Businesses may encourage their employees (financially) to use this system to reduce the number of cars and reduce their expenses on gas and transportation.

Table 1: SHi- A Technical Comparison

Metric	SHi (Proposed)	Traditional Ride-Sharing	Dynamic Ride-Sharing
Match Rate	High	Medium	High
Detour Distance	Low	Medium–High	Low
Response Time	Low	Medium	Medium–High
Matching Latency	Minimal	Moderate	Higher
Scalability	High	Very High	Medium

Table 1 presents a comparative-study of SHi with existing ride-sharing solutions based on technical metrics such as match rate, detour distance, response time, matching latency and scalability. SHi match rate is high due to the overlapping route optimization with auto matching support. In contrast traditional ride sharing route matching is medium because they rely on manual selection and availability. Dynamic ride sharing employs advanced matching algorithms to achieve high match rate high but its real world deployment remains limited. Detour distance of SHi and other dynamic ride sharing is low because both use optimization algorithms. In tradition ride sharing driver can deviate manually to achieve medium-high detour distance. Response time of traditional and dynamic ride sharing solution is medium as it depends on user browsing and computational complexity. Real time automated suggestion of SHi improves its response time. SHi matching latency is minimal due to instant system generated matches. Dynamic ride sharing also used optimization based algorithms to improve the matching latency while traditional ride sharing typically based on users driven decisions. Simulation based modeling makes dynamic sharing less scalable while SHi and tradition ride sharing are more scalable due to the employment of large cloud based systems and modular client server architecture respectively.

5.2 Future Functional Enhancements for SHi

SHi has a potential to grow as per the market and user needs. We have certain features and ideas in pipeline to enhance the current design of the proposed SHi solution

Voice Recognition: Imagine just telling your phone, “I’m driving from GULSHAN to NIPA at 5 pm with two open seats,” and it sets up the ride for you—no typing needed. This feature would make things way quicker and easier, especially if you’re on the go. Voice input is becoming more popular because it’s convenient and adding it, would make the app more accessible to everyone.

Account Ratings: To build trust, a system should have a bi-directional rating system where both drivers and passengers get ratings. If you’re a friendly, safe driver or a respectful passenger, you’d earn points and maybe get perks, like discounts from coffee shops along your route.

Account Credits & Expense Sharing: Driving costs money—gas, maintenance, etc. By allowing passengers to check these expenses, drivers could offer the ride amount without lengthy discussions. Its a win-win for both stakeholders; passengers get a cost-effective ride, and drivers get a bit of help with expenses. It’s like everyone pitching in to make the trip cheaper for all.

6. Data Privacy, Ethical Considerations, and System Limitations

SHi is user data centric approach as it uses the passenger data to provide them the best possible service. Our proposed system needs to follow the data privacy guidelines and ethical considerations to gain the client trust that their data is secure and no hidden breach is employed. SHi relies on real-time location sharing, route-tracking, and user personal data, so an inherited risk to expose the sensitive personal information is always raised a concern. We used certain control knobs to make SHi secure such as encrypted data storage and a controlled access to user data. Similarly, ethical points such as user agreement and consents, encapsulated data usage, strict user dashboard and only authorized tracking have been employed. Moreover, SHi clearly states and notifies the collection of user data along with a utilization mechanism.

With all these improved ride sharing features and algorithms SHi still has certain limitations which might restrict its wide adoptability. An evaluation in a very controlled environment may ignore the real world dynamics of users along with the ride sharing's actual complexities. Inaccuracies in GPS, variable traffic/network latency and actual user interaction and behavior with SHi pose serious concerns. And the most important, user willingness to share their ride has a great impact on match rate during less busy hours in a densely populated location. Catering all these limitations by employing a real world testing deployment will actually evaluate the SHi performance in real-time.

7. Conclusion

To overcome the traffic congestions and high travel costs, a dynamic and real-time ride-sharing solution **Smart Hitchhiking (SHi)** is designed. It optimizes urban commutes by real-time adaptation and secure data handling. SHi is equipped with modern and advanced features such as overlapping route identification, route correlation analysis and partial trip matching. An AVD emulator is used to test the functionality of the system. SHi offers an efficient and adaptive solution with user friendly GUI to facilitate the local users needs. Voice recognition, account rating, account credits and expense sharing are some of the future enhancements to improve the application. Along with the functional features enhancement discussed in the previous section we would also target quantitative metrics such as improvement in match rate, reduction in response time, and a consistent ride sharing service with atleast thousand concurrent user ride sharing requests. We also plan to evaluate SHi with real-world deployment and compare the obtained results with the controlled testing results

REFERENCES

1. Hasan, Abdulla Al-Towfiq. "Technology attachment, e-Attitude, perceived value, and behavioral intentions towards Uber-ridesharing services: the role of hedonic, utilitarian, epistemic, and symbolic value." *Journal of Contemporary Marketing Science* 5.3 (2022): 239-265.
2. Schaller, Bruce. "Can sharing a ride make for less traffic? Evidence from Uber and Lyft and implications for cities." *Transport policy* 102 (2021): 1-10.
3. Koling, Adam, et al. "Ride-Sharing the Wealth: Effects of Uber and Lyft on Jobs, Wages and Economic Growth." *Wages and Economic Growth* (2024).
4. Sun, Yanshuo, Shijie Chen, and Qianwen Guo. "Evaluating the environmental benefits of personalized travel incentives in dynamic carpooling." *KSCE Journal of Civil Engineering* 26.7 (2022): 3082-3093.
5. Pawlicz, Adam. "Pros and cons of sharing economy regulation. Implications for sustainable city logistics." *Transportation Research Procedia* 39 (2019): 398-404.
6. Shaheen, S., & Cohen, A. Shared ride services in North America: definitions, impacts, and the future of pooling. *Transport reviews*, *39*(4), 427-442. (2019).

7. Agatz, N., Erera, A., Savelsbergh, M., & Wang, X. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, *223*(2), 295-303. (2012).
8. “Backseat Surfing.” *VentureRadar*, www.ventureradar.com/organisation/Backseat%20Surfing/5d90653a-55a4-4aad-b2a5-3e54780761de. Accessed 5 Aug. 2025.
9. *Digihitch - Hitchwiki: The Hitchhiker's Guide to Hitchhiking*, hitchwiki.org/en/Digihitch. Accessed 5 Aug. 2025.
10. “Easy Lift Pk: Your Go-to Ride Sharing & Carpool Partner.” *EasyLift*, 6 July 2025, easylift.pk/.
11. *Humsafar - A Ride Sharing Application - Career Development Center, Cui Wah*, cdc.cuiwah.edu.pk/Event/Project/4376. Accessed 5 Aug. 2025.
12. Silwal, Shrawani, Md Osman Gani, and Vaskar Raychoudhury. "A survey of taxi ride sharing system architectures." *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2019.
13. Fielbaum, Andres, and Javier Alonso-Mora. "Unreliability in ridesharing systems: Measuring changes in users’ times due to new requests." *Transportation Research Part C: Emerging Technologies* 121 (2020): 102831.
14. Hussain, Zahid, Nabeel Ahmed, and Aftab Ali. "A Study on Consumer Satisfaction Toward Bykea e-Bike Services." *BOHR International Journal of Business Ethics and Corporate Governance* 1.1 (2022): 65-68.
15. Javaid, Ahson, Amna Javed, and Youji Kohda. "Exploring the role of boundary spanning towards service ecosystem expansion: A case of Careem in Pakistan." *Sustainability* 11.15 (2019): 3996.
16. Boduch, Adam. *React and react native*. Packt Publishing Ltd, 2017.
17. Matos, Victor. "Android environment emulator." *Cleveland State University* 20.11 (2009).
18. Bursztyn, Leonardo, et al. *Non-user utility and market power: The case of smartphones*. No. w33642. National Bureau of Economic Research, 2025.