Decentralized Ride Hailing platform based on Blockchain

Nilima Patil
Electronics and Telecommunication Engineering Department,
JSPM's Rajarshi Shahu College of Engineering,
Pune, Maharashtra, India
patilnilima774@gmail.com

Krishana Patil,
Electronics and Telecommunication Engineering Department,
JSPM's Rajarshi Shahu College of Engineering.
Pune, Maharashtra, India
krishnapatil644@gmail.com

Sahil Kuthe
Electronics and Telecommunication Engineering Department,
JSPM's Rajarshi Shahu College of Engineering,
Pune, Maharashtra, India
sahilkkuthe2004@gmail.com

Dr. S. A. Bhisikar,
Information Technology Engineering Department,
JSPM's Rajarshi Shahu College of Engineering.
Pune, Maharashtra, India
sabhisikar_entc@jspmrscoe.edu.in

Abstract—
The development and execution for a decentralized ride-hailing service that leverages the Solana blockchain to facilitate safe, affordable transactions is presented in this article. Web3.js, Next.js, and Rust-written smart contracts are used in the system's construction. Our method eliminates intermediaries by using blockchain-based smart contracts, in contrast to conventional ride-hailing systems that rely on centralized servers, which frequently result in exorbitant service fees, little transparency, and privacy concerns. This facilitates peer-to-peer transportation booking and payment transactions between users. The Google Maps application programming interface (API) is integrated into the program to offer route planning and real-time position monitoring. Web3.js manages the link between the user interface and the blockchain backend, while Next.js is used to construct the frontend for quick and seamless user interaction. This paper explains the system's architecture, the smart contract logic, and the flow of user interactions. We also evaluate the application's usability, scalability, and cost-effectiveness. The results show that decentralized technology can improve urban mobility by giving users more control, transparency, and financial advantages.

Keywords—Blockchain, Ride-Hailing, Solana, Rust, Web3.js, Smart Contracts, Next.js, Google Maps API

I. Introduction

Ride-hailing platforms like Uber and Ola have significantly improved urban transportation by offering easy and accessible services through centralized mobile applications. However, these centralized systems come with several limitations. Drivers are charged high commission fees, users have little control over how prices are set, and both riders and drivers face privacy concerns due to centralized data storage and processing.

Recently, blockchain technology has drawn interest as a possible remedy for these issues. It makes decentralized, transparent, and trustless systems possible, allowing users to communicate with one another directly without the need for middlemen.

We suggest a decentralized ride-hailing application based on the Solana blockchain to overcome the drawbacks of conventional ride-hailing systems. By eliminating the need for middlemen, this method aims to increase control for both drivers and passengers. The platform automates important procedures, including fare computation, payments, and ride confirmations, using Solana's Rust-based smart contracts. Solana is more suited for real-time applications like ride-hailing since it has quicker processing speeds and cheaper transaction fees than Ethereum-based solutions.

The user interface is created with Next.js, which helps deliver quick loading times and a responsive experience. To connect the frontend with the blockchain, Web3.js is used for tasks like linking digital wallets, verifying transactions, and accessing blockchain data.

This paper provides a detailed explanation of the system's architecture, development process, smart contract logic, and test results. Our goal is to show how decentralized technologies can help create more equitable, efficient, and privacy-friendly transportation solutions.

II. Algorithm & Software

Smart contracts created in Rust and implemented on the Solana blockchain are used by the decentralized ride-hailing platform to carry out necessary functions, including scheduling rides, matching drivers, figuring out fares, and handling payments [4]. The fundamental concepts and methods that underpin these features are described in this part, along with references to earlier research that influenced or supported this implementation.

3.1 Ride Request and Driver Matching Algorithm

To connect a rider with the most suitable driver, the system uses real-time geolocation data obtained through the Google Maps API. The process begins when a rider sends a ride request containing their current latitude and longitude. The system maintains a list of available drivers along with their locations. It calculates the distance from the rider to each driver using the Haversine formula and selects the driver who is geographically closest.

This method is inspired by similar approaches used in decentralized transport solutions [1][2].

Equation 1: Equation for Haversine Distance

$$\text{Cos}(\phi_1) \times \cos(\phi_2) \times \sin^2(\Delta\mu / 2) + \cos(\phi_1) = a$$

$$\texttt{Atan2}(\sqrt{a}, \sqrt{(1 - a)}) = c$$

$$R \times c = d$$

The rider and driver's latitudes (in radians) are denoted by ϕ_1 and ϕ_2 in the equation above, while the latitude and longitude discrepancies are denoted by $\Delta\phi$ and $\Delta\lambda$, respectively. R is the radius of the Earth, or around 6371 km, and d is the distance that results. This method guarantees equitable and ideal matching according to proximity [2][6].

3.2 Dynamic Fare Calculation Algorithm

The fare for a ride is determined based on multiple variables such as distance, a base fare, rate per kilometer, and a surge pricing multiplier. The system first calculates a base amount by multiplying the distance by the per-kilometer rate and adding it to the fixed base fare. Then, it applies a surge multiplier that adjusts according to the supply-demand ratio in real-time.

Equation 2: Final Fare Calculation

$$Fare = (B + (d \times r)) \times S$$

Here, *B* is the base fare (e.g., ₹30), *d* is the distance in kilometers, *r* is the cost per kilometer (e.g., ₹10/km), and *S* is the surge factor (e.g., 1.2× during high demand). This dynamic pricing model helps maintain service availability during peak times and discourages price manipulation, a known issue in traditional ride-hailing platforms [3][13][16].

3.3 Smart Contract-Based Payment Execution

To make the payment process secure and transparent, the platform uses Solana smart contracts that are triggered after the completion of a ride. The smart contract analyzes the rider's information, determines that the rider has sufficient balance, and then enables the transfer of money straight from the rider's wallet to the driver's wallet when the rider confirms trip completion through the user interface. A confirmation receipt is created and the transaction is permanently recorded on the blockchain.

An event named `RideCompleted` is emitted to record the status. This mechanism removes the need for intermediaries and helps build trust between users, aligning with principles established in earlier decentralized transport and payment platforms [5][6][8][15].

III. Methodology

This section outlines the step-by-step approach used in the design and development of the proposed decentralized ride-hailing application. The system is built using the Solana blockchain, smart contracts written in Rust, and Web3 technologies to ensure secure, transparent, and efficient interactions between drivers and riders. The development process is divided into five main stages: system architecture design, smart contract development, frontend integration, geolocation services, and final testing and deployment.

System Architecture Overview

Each of the four primary components of the system cooperate to give the user an uninterrupted experience.

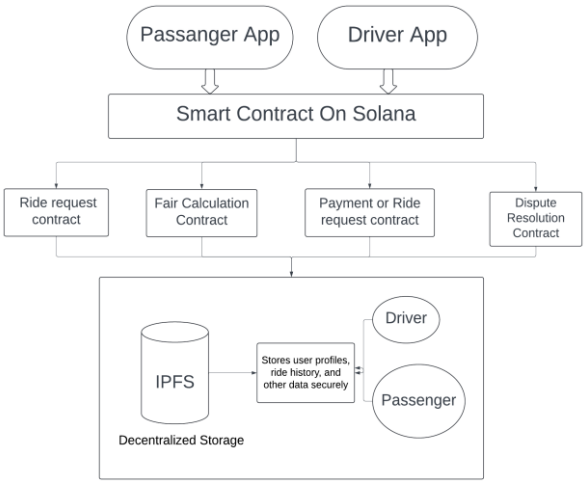
The frontend is developed using Next.js, offering fast rendering and responsiveness. It enables users to interact with the application through features like ride requests, wallet connections, and viewing ride history.

Using the Anchor framework, a smart contract layer is implemented on the Solana blockchain and coded in Rust.It handles core operations, including registering users, managing ride bookings, locking fares, and processing payments automatically once a ride is completed.

Web3.js is used to link the frontend with the blockchain. It manages wallet logins, executes transactions, and listens for blockchain events, helping the frontend stay in sync with the backend processes.

The system also incorporates geolocation services using the Google Maps API. This provides real-time tracking, route mapping, and fare estimation based on travel distance. A visual overview of the architecture is shown in Figure 1 of the paper.

Smart Contract Development



The smart contracts are at the heart of the application and are written in Rust. They are deployed on the Solana Devnet and carry out several important tasks. These include starting trip requests, registering drivers and passengers using their public wallet addresses, and locking the fare fee according to the frontend's distance calculation. The payment is immediately sent from the rider to the driver by the smart contract when a ride is over. It also emits events to keep the front frontend updated. All contracts are carefully tested using the Solana CLI and Anchor's testing tools before being deployed.

Frontend Development

To provide a sleek and contemporary design, the application's frontend is constructed with Next.js as well as stylized with Tailwind CSS. It supports secure login using Phantom Wallet, allowing users to sign in without a password. The rider interface enables input of pickup and drop-off points, while the driver dashboard shows available ride requests, ride progress, and transaction history. A transaction tracker shows real-time updates of blockchain activity. Web3.js is used to connect the frontend with the smart contracts, allowing for secure transaction signing and data exchange.

Geolocation Integration

Google Maps JavaScript API is used to add geolocation functionality to the application. It helps drivers and passengers visualize pickup and drop-off locations by providing an interactive map. The frontend estimates the price using the shortest route and total journey distance determined by the API. This fare information is then sent to the smart contract to be locked before the ride begins.

Tools and Technologies Used

The entire system is built using a combination of modern tools and technologies. These include Next.js for frontend development, Rust and the Anchor framework for writing and deploying smart contracts, Solana Devnet for blockchain deployment, Web3.js for blockchain interactions, Google Maps API for geolocation services, and Tailwind CSS for styling the user interface.

Technology/Tool	**Description**
Next.js	Framework for building the frontend UI
Tailwind CSS	Styling and responsive layout
Web3.js	Blockchain connectivity and wallet integration
Phantom Wallet	Solana-based wallet for user authentication
Rust + Anchor	Smart contract development on Solana
Google Maps API	Geolocation, route planning, and distance estimation
Solana CLI & Devnet	Smart contract deployment and testing

Feature	**Description**
Decentralization	No centralized entity controls user data or pricing
Security	All payments and operations are executed via verifiable smart contracts
Transparency	Transaction history is viewable on-chain using Solana Explorer
Cost Efficiency	Solana's low gas fees (~0.000005 SOL) reduce platform charges [4], [12]
Privacy	User identity is protected through wallet-based authentication without KYC
Scalability	Solana's throughput (up to 65,000 TPS) ensures the system is scalable [4]

The idea put forth builds a safe, effective, and expandable platform by utilizing blockchain and contemporary web

technology. Next.js is used as the main framework for building the frontend user interface, offering fast performance and server-side rendering capabilities. Tailwind CSS is integrated to ensure responsive design and consistent styling across devices. For blockchain connectivity and wallet integration, Web3.js is employed, allowing smooth interaction with the Solana network.

User authentication is handled through the Phantom Wallet, a popular Solana-based digital wallet. Smart contracts are built using Rust along with the Anchor framework, which offers a streamlined and effective approach to developing decentralized applications on the Solana blockchain. The use of the Google Maps API for features like distance estimate, route planning, and geolocation enhances the platform's usability. Solana CLI and Devnet are used for deploying and testing smart contracts in a development environment.

The system also benefits from key blockchain features. It supports decentralization by removing control from a single authority, giving users ownership over their data and transactions. Security is maintained through smart contracts, which automatically verify and execute operations. Transparency is maintained by making all transaction records accessible on the blockchain through tools like the Solana Explorer, allowing users to view and verify activity. The platform is cost-efficient due to Solana's low transaction fees, which significantly reduce the charges for users. Privacy is ensured by using wallet-based login methods that do not require Know Your Customer (KYC) procedures.

Lastly, Solana's architecture makes the system extremely scalable, enabling it to process as many as 65,000 transactions per second.

IV. Evaluation of the Literature

Sr. No	Ref. No	Publisher, Year	Technology	Benefits	Drawbacks
1	[1]	IEEE, 2020	PEBERS (The ride-hailing based on Ethereum)	Demonstrated feasibility of decentralized ride-hailing using Ethereum smart contracts	High gas fees, low scalability, latency due to Ethereum congestion
2	[2]	IEEE, 2022	Blockchain + IPFS for Ride-Hailing	Enhanced transparency and decentralized storage	Performance bottlenecks from Ethereum, IPFS integration complexity
3	[3]	IEEE, 2022	Incentive Auditing in Ride-Hailing	Prevents price discrimination through automated incentives	No real-time implementation, lacks usability evaluation
4	[4]	Solana Whitepaper, 2018	Solana Blockchain (PoH + BFT)	High throughput (65K TPS), Low latency, Minimal fees	Newer ecosystem, Limited mainstream adoption at the time
5	[5]	IEEE BigData Congress, 2017	Blockchain Architecture & Consensus	Detailed overview of blockchain design and trends	Generalized discussion, not specific to ride-hailing
6	[11]	Journal of Transportation Research, 2020	Blockchain in Transportation	Highlights privacy and transparency benefits	Conceptual, lacks implementation metrics
7	[12]	ICT4SD, Springer, 2022	Blockchain Cost Efficiency Study	Compared gas fees across platforms, validated Solana's low cost	Limited focus on live applications like ride -hailing
8	[13]	IEEE TMC, 2023	Driver Audit in Ride-Hailing	Enhanced trust and automated driver quality scoring	High computation cost, privacy preservation tradeoffs

V. Proposed System

The proposed decentralized ride-hailing system eliminates reliance on intermediaries by utilizing blockchain-based smart contracts to handle all core ride-related operations. It is built on the Solana blockchain, which manages transaction processing, while Google Maps API provides geolocation services. Web3.js facilitates blockchain interaction, and the frontend is developed using Next.js to offer a seamless and responsive user experience. The system is structured to promote transparency, cost-effectiveness, data privacy, and enhanced trust between drivers and riders.

The system architecture is composed of three primary layers. Both drivers and riders will find the Next.js-developed user interface layer to be easy to use. Riders can book rides, view routes, and authorize payments, while drivers can accept ride requests, check trip details, and track their earnings.

The core smart contract logic is written in Rust and implemented on Solana's Devnet utilizing the Anchor framework, which makes up the blockchain layer. Within this layer, smart contracts oversee critical functions such as registering riders and drivers, processing ride requests and matches, locking fares and managing payment releases, and logging events like ride initiation, acceptance, and completion.

The Google Maps API is integrated with Web3.js at the service layer. Invoking smart contract functions, maintaining wallet connections, listening to blockchain circumstances, and obtaining on-chain data are all handled by Web3.js. The Google Maps API makes it easier to design routes, enter locations, and determine how far the rider's pickup and drop-off locations are from one another.

A platform is composed of a number of useful components. The registration and authentication module enables riders and drivers to access the application using Phantom Wallet, with each wallet address serving as a unique identifier stored on-chain. This eliminates the need for conventional login credentials and ensures secure, decentralized authentication.

In the ride request module, a rider selects pickup and drop-off points through the map interface. This input generates a ride request transaction, which is submitted to the Solana blockchain, triggering the emission of a RideRequested event.

For the driver matching module, available drivers monitor the blockchain for new ride requests. Matching is conducted using the Haversine algorithm to determine proximity between drivers and riders. The smart contract updates the ride's condition and locks the fare whenever the driver accepts a request.

The fare estimation and locking module calculates ride costs using the formula: Fare = (B + d × r) × S, where B represents the base fare, d is the distance derived from the Google Maps API, r is the per-kilometer rate, and S is the surge pricing multiplier. The calculated fare is held within the smart contract until the trip concludes.

The driver uses the interface to certify the conclusion of the journey when the trip is complete. The ride status is confirmed by the smart contract, which also starts the money transfer from the rider's wallet to the driver's and sends out a RideCompleted event. Transparency and security are guaranteed by the permanent recording of all associated transactions on the Solana blockchain and their public accessibility.

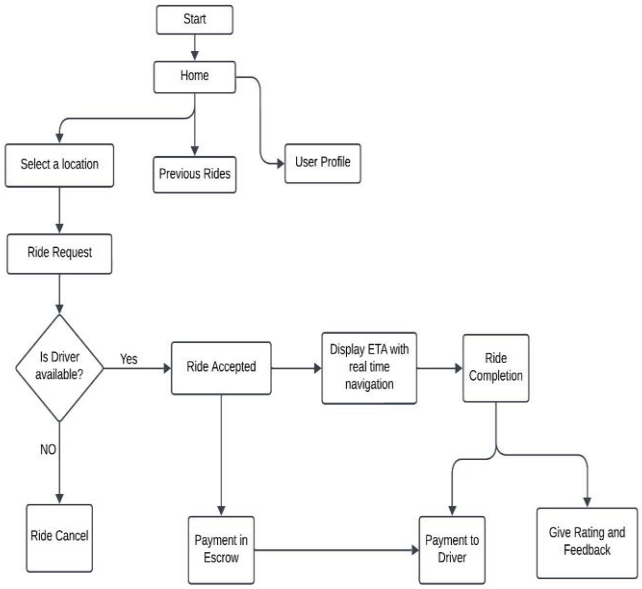


Fig. 3. Proposed System

VI. RESULT

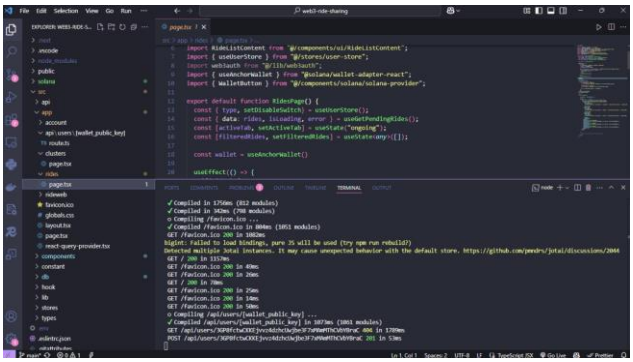


Fig 1. Project code

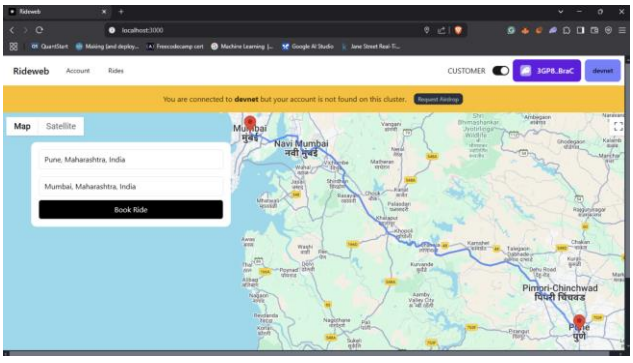


Fig. 2. Ride Booking

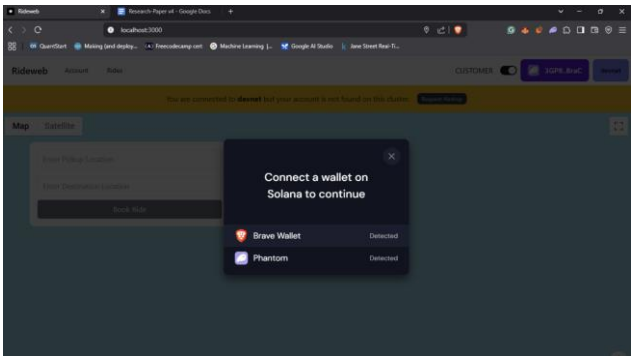


Fig. 3. Wallet Authentication options

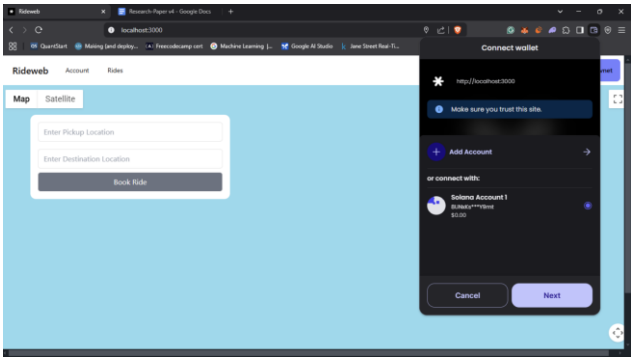


Fig 4. Wallet connection

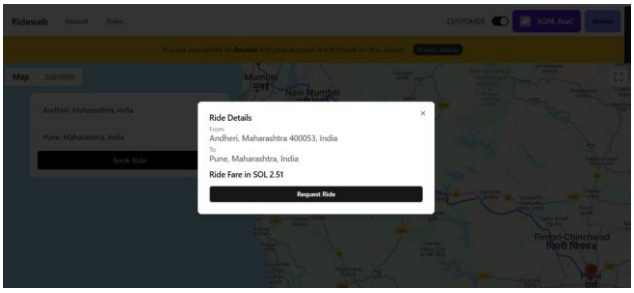


Fig. 5. Ride details

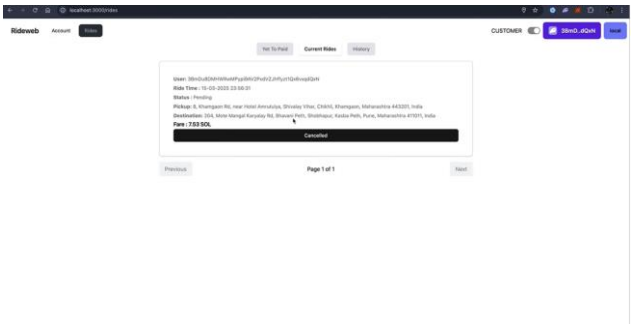


Fig. 6. Ride History

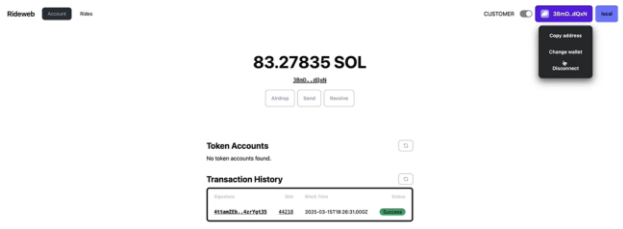


Fig 7. Transaction details

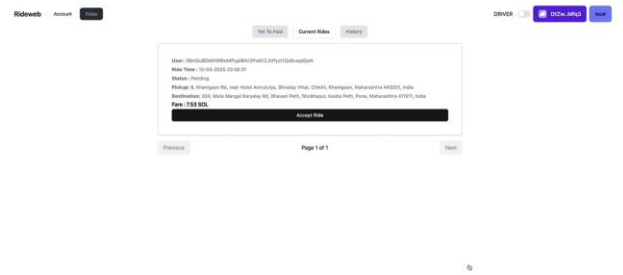


Fig 8. Driver Side

VI. Conclusion

This research introduces a decentralized ride-hailing application developed on the Solana blockchain, incorporating advanced web technologies such as Next.js, Web3.js, the Google Maps API, and smart contracts written in Rust. The platform aims to overcome common challenges found in traditional ride-hailing systems, including high commission rates, concerns over data privacy, and a lack of operational transparency. By facilitating direct interactions between drivers and riders through secure and immutable blockchain transactions, the application offers a more equitable and transparent alternative.

The implementation illustrates that decentralized ride-hailing is both feasible and efficient when built on the Solana network, which is known for its high throughput and minimal transaction fees. Core functionalities such as ride matching, fare estimation, and payment processing are managed through smart contracts, while geolocation and route tracking are handled using the Google Maps API. Additionally, the use of wallet-based authentication helps maintain user privacy without compromising security.

Testing was conducted on the Solana Testnet, yielding positive results in terms of usability, responsiveness, and cost-effectiveness on the blockchain. The proposed system architecture lays a solid groundwork for future exploration and real-world deployment of decentralized transportation platforms.

VII. Future Scope

The envisioned decentralized ride-hailing platform lays a strong foundation for transforming urban mobility, yet there are numerous avenues for further development and exploration. One key direction for future work involves deploying the platform on the Solana Mainnet to evaluate its performance in real-world scenarios with active users. Incorporating decentralized identity (DID) solutions could allow the creation of verifiable and privacy-preserving user profiles, effectively removing the reliance on centralized KYC procedures. The efficiency and customer satisfaction of the platform might be significantly increased by utilizing AI-driven algorithms for driver and passenger pairing and route optimization. The introduction of a native utility token could foster an in-app economy that supports user incentives, governance, and loyalty programs. To ensure data integrity and transparency, decentralized storage solutions such as IPFS or Arweave can be employed for storing ride history, user feedback, and related metadata in a secure and tamper-proof manner. Expanding compatibility to include various blockchain networks may also boost interoperability and offer users more flexibility. Furthermore, building mobile applications with cross-platform technologies like React Native can help attract a wider user base and accelerate the widespread adoption of decentralized ride-hailing services.

VIII. References

[1] S. Kudva, R. Norderhaug, S. Badsha, S. Sengupta, and A. S. M. Kayes, "PEBERS: Practical Ethereum Blockchain-Based Efficient Ride-Hailing Service," *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*, Doha, Qatar, 2020, pp. 56–61.

[2] O. Naik, N. Patel, S. A. Baba, and H. Dalvi, "Decentralized Ride Hailing System using Blockchain and IPFS," *2022 IEEE Bombay Section Signature Conference (IBSSC)*, Mumbai, India, 2022, pp. 1–6.

[3] Y. Lu, Y. Qi, S. Qi, Y. Li, H. Song and Y. Liu, "Say No to Price Discrimination: Decentralized and Automated Incentives for Price Auditing in Ride-Hailing Services," *IEEE Transactions on Mobile Computing*, vol. 21, no. 2, pp. 663–680, Feb. 2022.

[4] A. Yakovenko, "Solana: A New Architecture for a High Performance Blockchain v0.8.13," *Whitepaper*, 2018. [Online]. Available: https://solana.com/solana-whitepaper.pdf

[5] Z. Zheng, S. Xie, H. N. Dai, X. Chen, and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," *2017 IEEE International Congress on Big Data (BigData Congress)*, Honolulu, HI, USA, 2017, pp. 557–564.

[6] R. Shivers, M. A. Rahman, M. J. H. Faruk, H. Shahriar, A. Cuzzocrea and V. Clincy, "Ride-Hailing for Autonomous Vehicles: Hyperledger Fabric-Based Secure and Decentralized Blockchain Platform," *2021 IEEE International Conference on Big Data (Big Data)*, Orlando, FL, USA, 2021, pp. 3741–3750.

[7] N. Mahmoud, A. Aly and H. Abd Elkader, "Enhancing Blockchain-based Ride-Sharing Services using IPFS," *Intelligent Systems with Applications*, vol. 16, 2022, Art. no. 200135. [Online]. Available: https://doi.org/10.1016/j.iswa.2022.200135

[8] M. Li, Y. Chen, C. Lal, M. Conti, F. Martinelli and M. Alazab, "Nereus: Anonymous and Secure Ride-Hailing Service Based on Private Smart Contracts," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 4, pp. 2849–2866, Jul.–Aug. 2023.

[9] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[10] V. Buterin, "Ethereum White Paper," GitHub repository, 2013. [Online]. Available: https://ethereum.org/en/whitepaper/

[11] J. Smith and R. Brown, "The Potential of Blockchain for Transportation Systems," *Journal of Transportation Research*, vol. 45, no. 2, pp. 123–135, 2020.

[12] O. Mokalusi, R. Kuriakose and H. Vermaak, "A Comparison of Transaction Fees for Various Data Types and Data Sizes of Blockchain Smart Contracts on a Selection of Blockchain Platforms," in *ICT Systems and Sustainability: Proceedings of ICT4SD 2022*, Springer, Berlin, Germany, 2022, pp. 158–172.

[13] Y. Lu, et al., "Safety Warning! Decentralised and Automated Incentives for Disqualified Drivers Auditing in Ride-Hailing Services," *IEEE Transactions on Mobile Computing*, vol. 22, no. 3, pp. 1748–1762, Mar. 2023.

[14] H. Yu, H. Zhang, X. Yu, X. Du and M. Guizani, "PGRide: Privacy-Preserving Group Ridesharing Matching in Online Ride Hailing Services," *IEEE Internet of Things Journal*, vol. 8, no. 7, pp. 5722–5735, Apr. 2021.

[15] N. Szabo, "Smart Contracts: Building Blocks for Digital Markets," *Extropy: The Journal of Transhumanist Thought*, no. 16, 1996, pp. 28–37.

[16] R. Gaineddenova, "Pricing and Efficiency in a Decentralized Ride-Hailing Platform," *Working Paper*, 2021. [Online]. Available: https://doi.org/10.2139/ssrn.3740057

[17] M. Bez, G. Fornari and T. Vardanega, "The Scalability Challenge of Ethereum: An Initial Quantitative Analysis," *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, San Francisco East Bay, CA, USA, 2019, pp. 167–176.