Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

GETACAR: A Privacy-Preserving Platform for Ride-Pooling

Hans Hüppelshäuser

Course of Study:	M.Sc. Wirtschaftsinformatik
Examiner:	Prof. Dr. Marco Aiello
Supervisor:	Robin Pesl, M.Sc.
Commenced:	March 27, 2023
Completed:	Septeber 27, 2023

Abstract

The widespread adoption of autonomous vehicles is expected to lead to an overall increase in traffic. Ride-pooling can counter this downside of an otherwise promising technology, but the majority of current ride-pooling platforms utilise centralised designs that allow companies to collect vast amounts of user data. To solve this problem, we propose the decentralised ride-pooling platform GETACAR that focuses on privacy-preservation. GETACAR utilises blockchain technology to allow for the transparent and immutable tracking of ride processes without exposing personal information to other participants or the platform itself. To realise the platform, we develop its design, define its interactions and create a prototypical implementation. GETACAR is comprised of several components, including a customer and ride provider frontend allowing humans and autonomous vehicles to interact with GETACAR. We introduce an off-chain matching service to find the best possible match between customers and ride providers via a Vickrey auction. GETACAR also connects with crypto exchanges that allow the platform to use cryptocurrencies for internal transactions while users can still handle payments via fiat currencies. An authentication service verifies all parties wishing to participate on the platform, ensuring accountability and impeding the use of multiple identities. To ensure safety across the platforms, a robust rating system is in place that allows all parties to rate each other. In addition, a number of privacy mechanisms are in place to minimise the exposure of personal information, including location cloaking, pseudonyms, and frequently changing wallets. A prototype validates the GETACAR platform design by showcasing the platform's key features, including a fully realised user frontend, the matching service, and smart contracts running on the Ethereum blockchain. All these components are connected and working together, allowing for a customer to request a ride with multiple ride providers bidding on the ride. The implemented matching service determines the winner, and smart contracts manage the overall ride, including the rating process. Both the design of the platform and the prototype showcase the potential of blockchain technology to create next-generation ride-pooling platforms that ensure transparency while preserving privacy.

Kurzfassung

Es ist zu erwarten, dass die Verbreitung autonomer Fahrzeuge zu einem allgemeinen Anstieg des Verkehrsaufkommens führen wird. Ride-Pooling kann diesem Nachteil einer ansonsten vielversprechenden Technologie entgegenwirken, aber aktuelle Ride-Pooling-Plattformen nutzen zentralisierte Designs, die es Unternehmen ermöglichen, große Mengen an Benutzerdaten zu sammeln. Um dieses Problem zu lösen, setzt die dezentrale Ride-Pooling-Plattform GETACAR auf den Schutz der Privatsphäre. GETACAR nutzt die Blockchain-Technologie, um eine transparente und nicht manipulierbare Verfolgung von Fahrten zu ermöglichen. Die Plattform besteht aus einem Kunden- und Fahranbieter-Frontend, das es Menschen und autonomen Fahrzeugen ermöglicht, mit GETACAR zu interagieren. Es wird ein Off-Chain Matching-Service eingeführt, um über eine Vickrey Auktion die Kunden bestmöglich mit passenden Fahranbietern zusammenzubringen. Die Plattform ist mit einem Authentifizierungsdienst ausgestattet, der Pseudonyme generiert und Krypto-Wallets verifizieren kann, um sicherzustellen, dass Benutzeridentitäten nicht auf der Plattform preisgegeben werden. GETACAR ist mit Krypto-Börsen verbunden, die es der Plattform ermöglichen, Kryptowährungen für interne Transaktionen zu verwenden, während Benutzer weiterhin Zahlungen über Fiat-Währungen abwickeln können. Um die Sicherheit auf der Plattform zu gewährleisten, gibt es ein robustes Bewertungssystem, das es allen Parteien ermöglicht, sich gegenseitig zu bewerten. Ein Prototyp validiert das GETACAR-Plattformdesign, indem er die wichtigsten Funktionen der Plattform präsentiert, darunter ein vollständig realisiertes Benutzer-Frontend, den Matching-Service und Smart Contracts, die auf der Ethereum Blockchain laufen. Alle diese Komponenten sind miteinander verbunden und arbeiten miteinander, sodass ein Kunde eine Fahrt bei mehreren Fahranbietern anfordern kann und diese auf die Fahrt bieten können. Der implementierte Matching-Service ermittelt den Gewinner und Smart Contracts verwalten die gesamte Fahrt, einschließlich des Bewertungsprozesses. Sowohl das Design der Plattform als auch der Prototyp verdeutlichen das Potenzial der Blockchain-Technologie zur Schaffung von Ride-Pooling-Plattformen der nächsten Generation, die Transparenz gewährleisten und gleichzeitig die Privatsphäre wahren.

Contents

List of Figures

List of Tables

List of Listings

Acronyms

DApp Decentralised Applications. 24

DPoS Delegated Proof of Stake. 24

IPFS Interplanetary File System. 35

OSI Open Source Initiative. 42

PoS Proof of Stake. 24

PoW Proof of Work. 24

RSUs Road Side Units. 35

SUMO Simulation of Urban MObility. 88

1 Introduction

With autonomous vehicles expected to become a reality in the near future, an increase in overall traffic on the roads can be expected. The mass utilisation of ride-pooling platforms can counter this development, but the current ride-pooling platforms are built centralised and collect large amounts of sensitive user data [RGA+22]. Therefore, this research aims to create a decentralised, privacy-preserving ride-pooling platform that provides a feasible alternative to the current platform landscape.

1.1 Problem Statement

Technological breakthroughs in the area of autonomous driving have accelerated in recent years. While autonomous vehicles provide the ability to make travelling more convenient and efficient, they also can create problems regarding general traffic conditions [RGA+22].

The fast adoption of autonomous vehicles, once they enter the mass market, will inevitably result in more vehicles on the roads. This increase can be explained through the number of upsides that make autonomous vehicles appealing. Without the need to drive manually, more people will decide to travel with a personal autonomous vehicle and against public transport methods, creating stagnation and traffic jams. Therefore, while the technology promises several advantages, without proper intervention, it can also negatively affect travel [RGA+22].

Ride-pooling is one solution to tackle this problem. The concept centres around the idea of using a single vehicle to transport multiple passengers who all travel in a similar direction. Thereby, ride-pooling reduces the overall traffic of individual vehicles on the roads by improving the utilisation of seating capacity inside the vehicles. However, in practice, ride-pooling faces many challenges [RGA+22].

One of the big challenges with current ride-pooling platforms is the centralised nature of the available platforms. This centralisation allows companies to collect huge amounts of personal and transaction data, containing highly sensitive information like residential addresses, payment information, and travel habits that can be sold and utilised for targeted advertising. Having a single entity in control of the ride-pooling platforms also results in worse conditions for ride providers and higher prices for customers [BBA+21].

With this background, it becomes clear that a new approach towards ride-pooling platforms is needed to tackle the industry's problems and further popularise the concept of ride-pooling. This paper promotes the creation of a decentralised, trust-based ride-pooling platform which can tackle the problems of the current industry.

Blockchain technology has proven itself to be a capable tool for creating decentralised platforms in recent years. Platforms built with blockchain technology offer data immutability, transaction transparency, and decentralisation over a network of worker nodes by design. Blockchain technology addresses many of the problems inherent in centralised platforms. While the technology provides several advantages, it also brings with it several unique challenges regarding the privacy and anonymity of user activity on the platform, as all transactions running through the blockchain network are public by design [MAA22].

Therefore, this study takes up the challenge of conceptualising, designing, prototyping and evaluating a platform for ride-pooling based on blockchain technology that preserves privacy. This platform will bring customers and ride providers together to allow for seamless ride-pooling that ensures the privacy of transactions and creates an environment where parties only share necessary information with each other.

1.2 Objectives

To achieve the overall goal of this research, creating a decentralised ride pooling platform that provides an alternative to the centralised solutions, several research objectives have to be met:

Design of the Components and Interaction Flow between the Platform, Customer, and Ride Provider

The research needs to provide a design blueprint for the decentralised ride-pooling platform. This design should communicate the general vision of the platform and explain the key concepts. At its core, the platform is an ecosystem of components interacting with each other. Therefore, it is also necessary to design a streamlined, secure, and efficient flow for these interactions. It is also important to showcase how the individual components are deployed, especially in regard to the off-chain components that make up the platform. The objective here is to develop an interaction flow that ensures seamless ride booking, facilitates trustworthy transactions, and preserves privacy.

Design of a Trust Mechanism for Customer and Ride-Providers

Trust is a necessity for every platform but is especially relevant for decentralised platforms as these platforms are not managed by a single owner that can single-handedly settle disputes or resolve unexpected edge cases. Therefore, it is necessary for the platform to have a robust reputation system that sanctions malicious behaviour and promotes good behaviour.

Evaluation of Customer and Ride-Provider Anonymity and Privacy throughout the Platform

One of the disadvantages of blockchain-based platforms is that the high level of transparency can result in a neglect of customer and ride-provider anonymity and privacy. This counts especially for ride pooling platforms where large amounts of personal data like location and transaction data get exchanged. That is why it is important to assess the platform design regarding privacy and anonymity to show that no entity can collect critical amounts of data from the platform.

Proposal of Solutions for Physical Issues and Edge Cases

While the general focus of the research lies in creating digital processes that allow handling as much of the user flow through the platform as possible, it is important to also design solutions for potential damage to vehicles by passengers or inappropriate actions by individuals towards other passengers that need to be handled outside the platform. Therefore, the aim is to ensure accountability and conceptualise reporting mechanisms.

Prototypical Realisation of the Decentralised Platform

Based on the theoretical design, a prototype implementation of the platform is constructed. By building the platform, it is possible to simulate real-world scenarios, understand unforeseen challenges, and refine the design in response to them.

To ensure a realistic scope for this research, there are exclusions to some aspects of the platform. Therefore, this research will not cover creating a decentralised authentication service for the platform. The reason for this is that an authentication service is a generic component that is used for all kinds of decentralised platforms. However, the platform will discuss the general authentication flow that the platform will utilise. While ensuring the platform is generally economically feasible, this research will not cover the economic intricacies of running a decentralised ride-pooling platform. Through the defined objectives, the goal is to construct a platform that shows that the concept of a decentralised, privacy-protecting ride-pooling platform is feasible.

1.3 Methodology

It is important to utilise a structured approach when designing the decentralised ride-pooling platform. The following methodology provides a step-by-step process where each step builds upon the previous one, ensuring the platform viability, resulting in the design and prototypical implementation of a decentralised, privacy-preserving ride-pooling platform that showcases the current state of technology and scientific research in that field.

1. **Literature Research about Current Solutions**: First, it is important to understand the current state of scientific research. Therefore, this paper will conduct an in-depth analysis of the current state of scientific literature, reviewing academic papers, industry reports, and white papers about ride-pooling, decentralised systems, and related technologies. The output of this step is a comprehensive overview of what has been achieved in the area of decentralised ride-pooling so far.
2. **Identification of Shortcomings**: Building upon the previous stage, this research will work out potential shortcomings of the current research landscape, point them out and propose solutions to balance out these shortcomings. This allows for the decentralised ride-pooling platform built through this research to not only be a gathering of existing research findings but also to provide added value to the research landscape.
3. **Proposal of a Solution Design**: The next step is to create a comprehensive design for the platform based on the findings of the research analysis. This phase includes designing the architecture of the decentralised platform, defining interaction flows and outlining trust and privacy mechanisms.

4. **Prototypical Implementation of the Solution Design**: To prove the viability of the design, it is necessary to build a prototype that can showcase that the core functions and components work as intended. Therefore, this step includes the programming of smart contracts, user interfaces and other components and enabling them to communicate with each other. The finished prototype allows for real-world testing and iterative refinement.
5. **Evaluation Whether the Previously Set Requirements are met**: Based on the results from the working prototype, it is then important to analyse if all research objectives set for the decentralised ride-pooling platform are met. Based on this evaluation, it is possible to determine if the platform is a success and can provide a contribution to research.
6. **Identification of Limitations and Proposal of Possible Improvements**: The evaluation is also meant to bring up shortcomings of the research and aspects of the platform that require further investigation. These shortcomings and possible improvements are clearly pointed out so that future researchers can take the findings of this work and use it as the base for their research.

In summary, this iterative methodology ensures a scientific step-by-step approach to developing the privacy-preserving ride-pooling platform. The research approach, therefore, helps meet all research objectives set for this work.

2 Background Information

In the following chapter, we elaborate on relevant concepts and technologies that are necessary to understand for this research including autonomous driving, ride-pooling, blockchain and decentralised platforms. We additionally present an overview of the current state of academic literature in the field of decentralised ride-pooling platforms.

2.1 Autonomous Driving and Ride-Pooling

The advantages of autonomous driving, combined with the growing significance of ride-pooling, promise to have a strong impact on urban mobility [SKMM23]. We dive into the details of both concepts, exploring their origins, developments, and the potential synergy they hold for the future of transportation.

2.1.1 Autonomous Driving

Autonomous, or self-driving vehicles, combine hardware and software to navigate and control the car without human intervention [SKMM23]. Classified into levels 0 to 5, with five being fully autonomous, these vehicles rely on intricate systems of sensors, cameras, lidars, and radars. They continuously gather data about their environment, which is then processed by advanced algorithms to make driving decisions [HMS22]. Projects like the EUREKA Prometheus Project in the 1980s and the DARPA Grand Challenge in the early 2000s played essential roles in developing autonomous technologies [HMS22]. Today, significant tech and automobile companies compete to build fully autonomous vehicles for mass adoption [SKMM23]. The potential benefits of autonomous driving are vast:

Safety: Human error, responsible for most road accidents, could be drastically reduced [HMS22].

Efficiency: Optimal driving by autonomous cars might reduce traffic congestion and lead to more streamlined traffic flows [SKMM23].

Accessibility: Those unable to drive due to age, disability, or other factors can enjoy independent mobility [HMS22].

Economic Impact: A reduction in accidents implies decreased costs in healthcare and vehicle repairs [SKMM23].

However, challenges still exist. Technical complications, legal barriers, ethical questions (like decision-making in unavoidable accidents), and public scepticism must be addressed for a broader acceptance [HMS22].

2.1.2 Ride-Pooling

Ride-pooling allows for multiple people to share a single vehicle for a trip, where all passengers have different destinations but share a similar route [PHSB21]. Platforms like UberPool and Lyft Line have popularised ride-pooling in urban environments. The appeal of such services lies in their promise of reduced cost of travel for passengers, decreased overall traffic, lower carbon emissions, and the potential reduction of occupied parking spaces [Sha18]. However, ride-pooling is not without its challenges. Efficient route optimisation to ensure minimal detours, balancing demand and supply, and ensuring passenger safety are areas that ride-pooling providers struggle with [PHSB21].

The synergy of autonomous driving and ride-pooling offers a promising vision of the future of urban mobility [SKMM23]. Autonomous vehicles offer more efficiency while also reducing the overall cost of operation without the need for a human driver. This also allows for more overall vehicle space for additional passengers and cargo. In addition, autonomous vehicles can potentially work around the clock without downtime or increased prices for night trips [HMS22; SKMM23]. Regarding environmental impact, the combination of electric and autonomous vehicles, when integrated with ride-pooling services, allows for a reduction in environmental pollution [HMS22; SKMM23].

Autonomous ride-pooling also has an effect on city planning, as the design of modern cities is largely centred around vehicles moving and parking. Through a reduction of overall traffic and a sharp decline of needed parking spaces, large areas can be repurposed for housing, parks or recreational areas [SKMM23]. Lastly, at the core of these advancements lies an increase in accessibility. Autonomous ride-pooling systems lower barriers created by age, disability, or socioeconomic status and thereby allow large groups of society to participate in urban mobility that were previously excluded. [HMS22]. However, autonomous ride-pooling is not without potential downsides. Job losses, especially for human ride providers, the challenge of adjusting infrastructure to accommodate autonomous vehicles, and the need to build robust and safe systems are concerns that need to be addressed [HMS22].

In conclusion, autonomous ride-pooling platforms represent technological advancements and can change our approach towards transportation [Sha18], promising a more efficient, environmentally friendly, and inclusive transportation landscape [SKMM23]. However, the development of such platforms brings up the challenge of balancing the immense potential benefits with the inherent difficulties.

2.2 Blockchain Technology and Smart Contracts

The blockchain concept represents a technological breakthrough. The decentralised, immutable ledger at the core of the technology allows blockchains to be utilised in a number of industries, including finance and supply chain management, where the integrity and immutability of information play an important role [Zho23]. The decentralised consensus approach of blockchain ensures that data modifications are only possible through the unanimous approval of all participating systems. This ensures that information that is written onto the ledger becomes immutable. To provide all

the important information on blockchain for this research, we provide an overview of the current state of blockchain, explain the technical concepts behind blockchain in more detail, discuss the applications of smart contracts and take a closer look at the security of blockchain [TK22].

2.2.1 Introduction to Blockchain

Blockchain technology as we know it today originated with the introduction of Bitcoin in 2008 [Nak09]. Satoshi Nakamoto, the pseudonymous individual or group behind Bitcoin, introduced the concept as a solution to the double-spending problem in digital currencies [Nak09]. Digital currencies before Bitcoin faced the problem that it was very difficult to ensure that no token could be spent more than once. The solution to this problem presented by Nakamoto is a decentralised ledger, where every transaction gets verified by a network of nodes through a consensus mechanism [TK22]. This technological breakthrough was groundbreaking because it allowed the creation of decentralised currencies that are not controlled by a single entity like a government.

One of the outstanding features of blockchain technology is its decentralisation [GBE+18]. Unlike traditional databases, such as an SQL database operated by a central entity, blockchains operate on a peer-to-peer network [GBE+18]. Every participant (or node) has access to the entire database and the complete history of all transactions. This means that no single participant has control over the data, and all participants collectively maintain the integrity of the data.

Immutability is another critical feature of blockchains. Once a transaction is recorded on the blockchain, it becomes extremely difficult to alter [Pil16]. This is because each block contains a cryptographic hash of the previous block, creating a chain of blocks [Pil16]. To change a single block, one would need to alter all subsequent blocks, which is computationally impractical, especially in large networks [CSJ+17].

Transparency is inherently built into the system due to its open-source nature. Every transaction on the blockchain is visible to anyone who chooses to view it, ensuring full transparency in the network [BKB21]. However, personal information about the users conducting the transactions remains private as each user is commonly represented through some form of Public Key [Wei22]. This ensures a balance between transparency and privacy.

While the foundational principles of blockchains remain consistent, there are different types tailored to specific needs [GBAC21]. Public blockchains, like Bitcoin and Ethereum, are open to anyone and are simply secured by their cryptographic algorithms [GBAC21]. In contrast, private blockchains, like the Hyperledger Blockchain Projects, can be restricted to a specific group of participants, often used by businesses for internal processes. Private Blockchains can also be used as consortium blockchains or federated blockchains, operated under the leadership of a group [LPZ+23]. They provide a balance between the openness of public blockchains and the restrictions of private ones.

2.2.2 How Blockchain Works

Diving deeper into the mechanics of blockchain technology shows the interplay of cryptographic principles, network theory, and consensus algorithms [XCW+22]. At the core of this technology are blocks, which are essentially records of transactions. Each block typically contains a timestamp, a reference to the previous block (known as the parent block), and a list of transactions. These

transactions are represented as cryptographic hashes, which are fixed-size strings of characters generated from input data of any size. The advantage of these hashes is that even a small change in the input data results in a completely different hash, ensuring the integrity of transaction records that can be traced back to the very first block, known as the genesis block [XCW+22].

Central to the operation of a blockchain is the concept of consensus mechanisms [TSA22]. These are protocols that ensure all participants in the network agree on the validity of transactions. The most well-known consensus mechanism is Proof of Work (PoW). In PoW, participants, often referred to as "miners", solve complex mathematical problems to validate transactions and create new blocks. This process requires significant computational power and energy. An alternative mechanism, Proof of Stake (PoS), determines the creator of a new block based on their stake or ownership of the cryptocurrency. It is seen as a more energy-efficient alternative to PoW. Delegated Proof of Stake (DPoS) further refines this by allowing coin holders to vote for a few trusted nodes to validate transactions, streamlining the process and reducing the energy footprint [KUC21].

The blockchain network is maintained by nodes, which are computers participating in the network [XCW+22]. In general, there are two primary types of nodes: full nodes and light nodes [MTD21]. Full nodes store the entire blockchain and validate all transactions and blocks. They serve as the network's backbone, ensuring data integrity and consistency. Light nodes, on the other hand, store only a subset of the blockchain and rely on full nodes for transaction validation and other heavy operations [MTD21]. Their primary role is facilitating faster and more efficient interactions with the blockchain.

Transactions allow users to interact with the blockchain. Once a transaction is initiated, it is broadcast across the blockchain network and placed in a pool of unconfirmed transactions. Worker nodes then take these transactions from the pool and validate them against the ledger history to ensure that they are valid. If a transaction is determined as valid, it is placed in a block, together with other valid transactions. Once the block is full, it is shared with the network for verification through the consensus mechanism. After this, the block is added to the chain, and the transaction becomes a permanent part of the ledger history [XCW+22].

2.2.3 Smart Contracts

As one part of blockchain technology, smart contracts have emerged as an advanced tool, extending the use-cases of blockchains beyond the record keeping of transactions [UX23]. A smart contract is a self-executing contract where the terms of agreement or conditions are represented through written lines of code [ZMM22]. They are protocols that verify and enforce credible transactions without the need for third parties [ZMM22]. At their core, they are digital contracts that automatically execute actions when predefined conditions are met.

The concept of smart contracts is not new, but its practical application gained popularity with the advancements of blockchain technology [Pie21]. Ethereum, launched in 2015, demonstrated the potential of smart contracts [Pie21]. Ethereum's platform is designed specifically to create and execute smart contracts, providing a more flexible scripting language and a platform for creating a Decentralised Applications (DApp) [Pie21]. Since Ethereum's creation, a number of other blockchains have integrated smart contract capabilities, offering unique features and optimisations.

Once deployed, smart contracts operate without human intervention, ensuring that transactions are carried out correctly when conditions are met [UX23]. This allows for interactions among parties that are not required to trust each other. Since the contract is on a blockchain, all parties can verify the contract's code and monitor its execution [UX23]. The decentralised nature of blockchains also ensures that smart contracts are secure from tampering, providing an added layer of security [ZMM22].

Understanding the life cycle of a smart contract provides insights into its operational model. The journey begins with its creation, where the contract's terms are defined and encoded. Once the code is written and tested, it is deployed onto the blockchain, appearing as an immutable part of the ledger [TK22]. After the deployment, the contract is now active and can start receiving and processing information. Execution occurs when the conditions specified in the contract are met, triggering the actions encoded in the contract [Pie21]. While many smart contracts are designed to run without a predefined end, there are scenarios where they might have a termination condition, ending the contract's active state on the blockchain [TK22].

Even though smart contracts have many advantages, they also come with their own set of limitations and challenges [Nzu19]. One notable challenge in the Ethereum network is the concept of Gas fees. Every operation, from contract deployment to execution, requires computational resources. Users pay for these resources using so-called Gas, and with increased network activities, these fees can rise. Scalability remains a concern as well. As more complex smart contracts and DApps are developed, there is a growing demand for blockchains to process more transactions per second without compromising on security or decentralisation [TK22]. Lastly, smart contracts are only as good as the code they're written in. Coding errors or oversights can lead to vulnerabilities, potentially allowing malicious actors to exploit the contract [ZMM22].

In conclusion, smart contracts represent a significant leap in how agreements and transactions can be managed on a decentralised network [Nzu19]. While they offer many advantages, it is necessary to consider their challenges to utilise their full potential [Nzu19].

2.2.4 Blockchain Security

Blockchain's decentralised nature, which is often used for its resilience and transparency, also presents unique security challenges. One of the most discussed vulnerabilities is the 51 Percent attack. In such an attack, if a single entity gains control of more than half of the network's nodes, it can potentially double-spend coins and stop or reverse transactions. This undermines the trust and integrity of the blockchain. Similarly, Sybil attacks occur when a single party controls multiple nodes, aiming to flood the network with false transactions or undermine mechanisms that rely on redundancy and trust [SHY21].

Smart contracts have their own set of security concerns [ANWK21]. Reentrancy attacks are a prime example, where an attacker drains funds from a contract by repeatedly calling its functions before the initial function call is completed [ANWK21]. Issues like overflow and underflow, where variable values exceed their set limits, can also be exploited, leading to unintended consequences in contract execution [GCZ+22]. These vulnerabilities underscore the importance of rigorous code audits and testing before deploying smart contracts on a live network.

Most blockchains offer pseudonymity, where transactions are linked to a cryptographic address rather than personal identities. However, through analysis, patterns can emerge, potentially de-anonymizing users [KL22].

In essence, while blockchain offers robust security mechanisms at the core of its design, it is not impenetrable. As the technology matures, addressing these vulnerabilities will be an important part of ensuring its general adoption and trustworthiness.

2.3 Decentralised Platforms and Data Privacy

Decentralised platforms, at their core, are systems where components, like resources or operations, are not controlled or managed by a single, central entity. Instead, they are distributed across multiple nodes, with each having equal authority and autonomy. This is in direct contrast with traditional centralised systems, where a single entity or a group of entities holds all the power and control. In the following chapter, we will introduce the core concepts behind decentralised platforms and showcase how data privacy can be handled in a system without a central authority.

2.3.1 Introduction to Decentralised Platforms

One of the primary characteristics of decentralised platforms is that it does not have a central point of control. This means that no single entity has the authority to make decisions on their own or changes without consensus from most of the network's participants. This leads to enhanced security, as the absence of a single point of failure makes the system more resilient to attacks [MLC+22]. Additionally, decentralised platforms often employ cryptographic techniques to ensure data integrity, privacy, and authentication. This ensures that transactions and interactions on the platform are secure, verifiable, and immutable.

Comparing decentralised platforms with centralised systems reveals strong differences in their operational philosophies. Centralised systems, such as traditional databases or web servers, are controlled by a single entity. This central authority has the power to set rules, make changes and grant or deny access [MLC+22]. While this centralisation can lead to efficiencies in terms of decision-making and simpler system architectures, it also presents vulnerabilities. A single point of failure in a centralised system can lead to the entire system collapse. Moreover, centralisation often results in data silos, where a single entity has control over large amounts of data [RMP+22].

On the other hand, decentralised platforms operate on the principles of democracy and transparency. Decisions are made based on consensus algorithms, ensuring that no single participant can dominate or manipulate the system. This democratisation of control can result in trust among users, as the platform operations are transparent [HSY+22]. Data in decentralised systems is typically stored across multiple nodes, ensuring redundancy and resilience. Even if one or more nodes fail, the system can continue to operate seamlessly [RMP+22].

There are several potential benefits for decentralised platforms. Firstly, they offer enhanced security and resilience due to their distributed nature. The risk of system-wide failures or attacks is significantly reduced [MLC+22]. Secondly, they promote transparency and trust among users, as

decisions are made collectively and openly. Additionally, decentralised platforms lead to innovations in peer-to-peer transactions, like smart contracts and decentralised applications, that allow for new business models [RMP+22].

However, decentralised platforms come with a set of challenges [HSY+22]. The lack of a central authority can lead to slower decision-making, as achieving consensus can be time-consuming. Additionally, the technology enabling decentralised platforms, such as blockchain, is still maturing, leading to scalability and performance issues. Interoperability between different decentralised platforms is also a challenge [Zha22].

2.3.2 Data Privacy: Definition and Importance

Data privacy, at its core, refers to the right of individuals to control or influence what information about them is collected and how it is used. It centres around the rules put in place to protect personal information and ensure that individuals remain in control of it [CSFS20]. This concept is crucial for several reasons.

Data privacy is directly linked to personal autonomy and dignity. Personal data can reveal intimate details about an individual's life, preferences, and habits. Ensuring that such information is not misused or mishandled is vital [CSFS20]. Without robust data privacy measures, individual rights can be violated, leading to a loss of trust in digital systems and platforms.

Furthermore, in the context of businesses and services, data privacy is important for maintaining consumer trust. Companies that fail to protect user data or misuse it can face significant reputational damage, legal consequences, and financial losses [LWW+19]. In sectors like decentralised ride-sharing, where users share location data, payment details, and personal preferences, ensuring data privacy can be the difference between a successful platform and one that users abandon due to trust issues [LWW+19].

While data privacy is a critical concept, it is essential to differentiate it from related terms like data security and data protection, as they are often used interchangeably but have distinct meanings. Data security refers to the protective measures and technologies used to secure data from unauthorised access. It focuses on defending data from malicious threats, like hackers, malware, or other cyber-attacks. For instance, using encryption to secure data to prevent unauthorised access is an example of data security practices.

On the other hand, data protection is a broader concept that includes both data privacy and data security. It refers to the policies, procedures, and legal measures designed to ensure that data is collected, stored, and used in a way that respects individual rights and complies with relevant laws and regulations [CSFS20].

In conclusion, data privacy is the right of individuals to control their personal information and its usage [CSFS20]. As we continue to integrate digital platforms into our everyday life, understanding and prioritising data privacy will become even more important.

2.3.3 Mechanisms of Data Privacy in Decentralised Blockchain Systems

Decentralised blockchain systems are a revolutionary technology offering data transparency, immutability and security. However, the nature of public blockchains, which are open and transparent, creates significant privacy challenges. Every transaction and its associated data are visible to anyone who accesses the blockchain, leading to potential privacy breaches and exposure of sensitive information.

Encryption plays an important role in addressing these challenges. At its core, encryption involves converting data into a code to prevent unauthorised access. In the context of blockchains, wallets consisting of a public key (often linked to an address on the blockchain) and a private key are widely used. A public key, visible to everyone, is used to encrypt data, while a private key, known only to the owner, is used to decrypt it. This ensures that only the intended recipient can access the information. Furthermore, end-to-end encryption ensures that data remains encrypted during its entire journey from the sender to the recipient, preventing potential eavesdroppers from accessing the information during transmission [TK22].

One approach to establishing an encrypted connection over a public network is the Diffie-Hellman Key Exchange. The Diffie-Hellman Key Exchange, introduced by Whitfield Diffie and Martin Hellman in 1976, is a cryptographic protocol that allows two parties to independently generate a shared secret key over an insecure communication channel. The protocol is based on the mathematical properties of modular arithmetic and discrete logarithm problems. Specifically, given a prime number p and a base g (where g is a primitive root modulo p), each party selects a private key and computes a public key. The public keys are then exchanged, and each party uses the other's public key along with their own private key to compute the shared secret. The security of the protocol relies on the difficulty of the discrete logarithm problem: while it is computationally easy to generate the public key from the private key, the reverse operation is considered infeasible with current technology when large prime numbers are used [DH76].

Another advanced cryptographic technique employed in blockchains is zero-knowledge proofs (ZKPs). ZKPs allow one party to prove to another that a statement is true without revealing any specific information about the statement itself. For instance, in a transaction, a user can prove they have sufficient funds without revealing the exact amount. This ensures transaction validity while preserving user privacy [TK22].

Homomorphic encryption offers another layer of privacy. It allows computations to be performed on encrypted data without first decrypting it. The result, when decrypted, remains accurate. This means that blockchain systems can process transactions and maintain data integrity without exposing the actual data, a boon for privacy-centric applications [PDKJ22].

While public blockchains offer transparency, they create challenges for the development of applications, especially those requiring higher levels of privacy. Private and consortium blockchains emerge as alternatives in such scenarios. Private blockchains restrict participation to selected entities, while consortium blockchains involve multiple organisations governing the network. Both these types limit data visibility to only authorised participants, enhancing data privacy [NDT20].

Off-chain storage is another solution to the privacy challenges. Instead of storing all data on the blockchain, only essential information is kept on-chain, while the rest is stored off-chain in secure databases. This reduces the amount of data exposed on the public ledger, ensuring privacy [YTH+22].

Lastly, layer 2 solutions, built on top of the primary blockchain, offer scalability and privacy improvements. By processing transactions off the main chain and only writing the final state on-chain, these solutions can ensure faster transactions and enhanced privacy [SKK+22]. In conclusion, while decentralised blockchain systems present certain inherent privacy challenges, a combination of cryptographic techniques and architectural solutions can effectively address these concerns.

2.4 Current Solutions and Shortcomings

Due to the diverse literature regarding decentralised ride-pooling platforms, the proven approach of a systematic literature search according to Vom Brocke was chosen [vSN+09]. In this way, quality criteria such as traceability and reproducibility can be ensured through a clearly defined process. Two common cross-publisher research databases and one common publisher database were used for the literature search. The selection of several cross-publisher research databases is intended to ensure that the search provides a representative overview of existing research on decentralised ride-pooling platforms. The selection of the database of a publisher with a focus on information technology is intended to show how the research topic is treated in the literature from a primary information technology perspective. The cross-publisher research databases used are Scopus and Ebscohost. The publisher database is IEEE Xplore. The goal is to obtain research literature as a search result that deals with the development of decentralised ride-pooling platforms. To obtain results covering mainly decentralised platforms, the search phrase “decentralised” was used. The following three synonyms were used to obtain search results that deal with the topic of ride-pooling: “ride-sharing”, “ride-pooling” and “ride-hailing”. Initial tests have shown that results with this search phase return suitable research papers without noticeable gaps in regard to the topics covering decentralised ride-pooling.

The complete search phrase looks as follows:

```
("decentralised" AND ("ride-pooling" OR "ride-sharing" OR "ride-hailing"))
```

For Scopus, Epscohost (all selectable databases included) and IEEE Xplore, the search phrase was applied to the title, abstract and keywords of the publications. Initial tests have shown that restricting the search to title, abstract, and keywords is the best compromise between the quantity and quality of the search results. Only literature that was published after 2014 (2015 – 2023) was considered for the literature search. This is to ensure that the specialist literature found is of current relevance without overly restricting the scope of the existing research literature. Likewise, after the initial compilation of the search results, all duplicates were removed. In this way, it is avoided that publications are counted twice because they are listed in several literature databases.

2.4.1 Selection of the findings

The literature search was carried out between the 28. July and the 18. of August 2023, resulting in 86 hits. A criteria-based selection was made beyond the search phrase and the time limit for the publication of the specialist literature. The exclusion criteria used in the criteria-based selection are no publications in languages other than English, no panels and comments, and no literature dealing

with decentralised platforms or ride-pooling. In addition, publications that are not freely available or accessible via a license from the University of Stuttgart had to be excluded. The inclusion criteria used are only publications in English, only publications from 2015 onwards, and only papers discussing the technical development of decentralised ride-pooling platforms [BFG+15]. Following Bandara, a first check of the actual relevance of the hits for answering the research question was carried out by screening the title, keywords and abstract. A full-text analysis was then carried out on the literature that was still considered relevant after the initial screening. Applying the inclusion and exclusion criteria in the initial screening and the subsequent full-text analysis, 10 relevant publications were identified from the 86 search hits for answering the research question. Additionally, two more relevant papers could be identified by following citations from the relevant literature. Table 2.1 shows how the relevant research literature is distributed across the research databases.

Table 2.1: Results of the Literature Search

Scientific Database	Search Results	Excluded Literature	Included Literature
Scopus	54	49	5
Epscohost	2	2	0
IEEE	30	25	5
Citation search			2
Total	86	76	12

The analysis of the publications shows that many different approaches are discussed in the scientific literature on how decentralised ride-pooling platforms can be built. For the results of the literature analysis to be evaluated and interpreted, the results must first be structured. For this purpose, a concept matrix approach, according to Webster and Watson, is pursued [WW02]. Based on the concept matrix approach, the specialist literature identified as relevant is assigned to eight topics relevant to creating a decentralised ride-pooling platform. These eight topics are derived from a general analysis of the topics covered by the scientific literature combined with topics relevant to fulfilling the research objectives:

Blockchain Utilisation: Blockchain is the underlying technology used for the creation of the decentralised ride-pooling platform. The literature needs to show in detail how blockchain technology is utilised by smart contracts and cryptocurrencies to build a ride-pooling platform.

Payments and Service Fees: The decentralised ride-pooling platform must manage ride payments and general service fees. Therefore it is important for the literature to show how these financial transactions can be implemented and how to ensure that ride providers are compensated fairly for their services inside the decentralised ride-pooling platform.

Privacy and Anonymity: Using blockchain technology demands a robust architecture that ensures privacy and anonymity for all users inside the platform. The scientific literature must showcase how users can engage with the platform and other users without revealing their identity directly or implicitly by sharing too much personal data with the platform over a longer time period.

Security and Resilience: For a decentralised platform to gain widespread adoption, it must guarantee the safety and security of all parties. While the blockchain itself already provides many security features by design, it is important for the literature to show how the off-chain components are hardened and how to prevent the off-chain components from providing false information to the on-chain components.

Trust Mechanisms: As decentralised platforms cannot rely on a central trusted authority, robust trust mechanisms become essential. The research papers must explain how community trust mechanisms can be successfully implemented into a decentralised platform.

Off-Chain Edge Cases: It is impossible to handle every edge case through the decentralised platform. As there is no central authority, it is important to provide alternative solutions to solve these problems without contradicting the decentralised nature of the platform. The research needs to recognise the existence of these edge cases and has to provide solutions to handle them.

Customer and Ride Provider Interaction Flow: The customer and ride provider interaction flow stands in the centre of the decentralised ride-pooling platform. The literature needs to provide insights into how this flow should look to utilise the advantages of blockchain technology.

Prototypical Realisation: Before building a market-ready version, the decentralised ride-pooling platform should be built as a prototype that showcases the most important aspects of the platform and proves its feasibility. Therefore, it is important for the literature to include a prototypical realisation of the platform that provides important insights that cannot be derived from the architecture alone.

As a result, the concept matrix shows the frequency with which the concepts dealt with in the specialist literature are distributed over the nine topics of decentralised ride-pooling. The assignment of the concepts on the x-axis to authors of the relevant specialist literature on the y-axis can be seen in Table 2.2. If a research paper covers a topic in detail, it is marked with ✓ ✓. if a research paper covers some aspects of a topic, it is marked with a ✓ . If a paper does not cover a topic at all or in a way that does not align with the objectives of this research, it is marked with a ×.

Table 2.2: Literature Search Matrix

Research Paper / Topic		Blockchain Utilisation	Customer and Ride Provider Interaction Flow	Payments and Service Fees	Privacy and Anonymity	Security and Resilience	Trust Mechanisms	Off-Chain Edge Cases	Prototypical Realisation
B-Ride: Ride Sharing With Privacy-Preservation, Trust and Fair Payment Atop Public Blockchain	[BLM+21]	✓ ✓	✓ ✓	✓	×	×	✓	×	✓
Application of Blockchain Technology to Smart City Service: A Case of Ridesharing	[CC19]	✓	✓	×	✓	✓ ✓	×	×	×
Ride-Hailing for Autonomous Vehicles: Hyperledger Fabric-Based Secure and Decentralize Blockchain Platform	[SRF+21]	✓	✓	×	✓	×	×	×	✓
RiderS: Towards a Privacy-Aware Decentralized Self-Driving Ride-Sharing Ecosystem	[BFJ20]	✓	✓	✓	✓✓	✓	×	×	✓
A Decentralized Ride-Hailing Mode Based on Blockchain and Attribute Encryption	[ZZHH22]	✓	✓	✓ ✓	✓	×	×	×	✓
Enhancing Blockchain-based Ride-Sharing Services using IPFS	[MAA22]	✓✓	✓✓	✓	✓	✓	×	×	✓
BlockWheels - A Peer to Peer Ridesharing Network	[JSD+21]	✓	✓✓	✓	✓	×	×	×	×
A Light Blockchain-Powered Privacy-Preserving Organization Scheme for Ride Sharing Services	[BMS+20]	✓	✓ ✓	✓	✓	×	×	×	×

Continued on next page

Table 2.2 – continued from previous page

Research Paper / Topic		Blockchain Utilisation	Customer and Ride Provider Interaction Flow	Payments and Service Fees	Privacy and Anonymity	Security and Resilience	Trust Mechanisms	Off-Chain Edge Cases	Prototypical Realisation
BlockV: A Blockchain Enabled Peer-Peer Ride Sharing Service	[PR19]	✓	✓	✓	×	✓	✓	✓	✓
Blockchain-Based Ride-Sharing System with Accurate Matching and Privacy-Preservation	[BBA+21]	✓	✓	×	✓	✓ ✓	×	×	×
Towards Blockchain-based Ride-sharing Systems	[VL21]	✓ ✓	✓	✓	✓	✓	×	×	✓
Co-utile P2P ridesharing via decentralization and reputation management	[SMD16]	✓	✓	✓	✓ ✓	✓	✓	×	×

2.4.2 Scientific Literature findings

The concept matrix 2.2 shows that the literature review did not identify a single paper that provides detailed coverage of all topics and would thereby allow us to answer all research objectives. The matrix also shows that while many of the papers discuss multiple topics, they often remain on a conceptual level without the goal of developing a feature-complete platform. It is still very important to take a detailed look at the identified literature to discuss their approaches to developing a decentralised ride-pooling platform. In the following, we will take a look at the outstanding features that are proposed in each paper and evaluate how they can support the creation of our feature-complete ride polling service.

Akbar et al. introduce B-Ride, a decentralised ride-sharing service built on public Blockchain [BLM+21]. B-Ride ensures ride data privacy for both drivers and riders. To counter malicious users exploiting the blockchain's anonymity, the system introduces a time-locked deposit protocol using smart contracts and zero-knowledge set membership proof. This ensures trust and commitment from all participants. A unique "pay-as-you-drive"methodology is proposed for fair payment, where drivers are compensated based on the distance covered. This system has many advantages. It ensures that the ride provider gets paid for the driven distance, and the customer does not have to deposit more money than necessary at once. The problem with this approach is,

that it requires a so-called Location Prover. These hardware devices ensure that the car provides honest location information about its position. While this technology is superior to systems that do not rely on Location Prover, a global network of Location Provers is currently not feasible. Therefore our platform will utilise an upfront deposit of the expected ride cost by the user that can be claimed by the ride provider after completing the ride. Additionally, B-Ride features a decentralised reputation management mechanism, rating drivers on past behaviour and incentives them to maintain good conduct. The system was successfully implemented and tested on the Ethereum blockchain, highlighting its real-world applicability. While a rating system is needed to ensure trust on the platform, B-Rides implementation also relies on Location Provers. Therefore, we will look at other research papers and their approaches in regards to rating mechanisms.

The authors Chang and Chang, highlight in their research paper the challenges faced by traditional ridesharing platforms [CC19]. To address the challenges of traditional ridesharing platforms, the SmaRi system leverages blockchain technology and smart contracts. This approach not only ensures secure and automated transactions but also promotes decentralised decision-making. The research emphasises the potential of blockchain in reshaping ridesharing services. A notable design decision by the authors is to use an off-chain authentication service called social networking service. This service allows users to utilise social media accounts to share rides with friends and to authenticate against the platform. While this concept is not covered in depth it provides insides into the many advantages of an off-chain authentication service.

Shivers et al. address the problems of centralised ride-sharing platforms. The authors propose a decentralised approach using blockchain technology, allowing individual autonomous vehicle owners to contribute their vehicles to a community-driven fleet when not in use. [SRF+21] The chosen blockchain platform for this endeavour is Hyperledger Fabric. The paper is notable for utilising a private blockchain to tackle the problems regarding anonymity and privacy, which are inherent downsides of public blockchains. The decision between a public and a private blockchain is one of the core architectural decisions for our own ride-sharing platform. After taking the arguments by [SRF+21] as well as other research papers into consideration, we decided to go forward with a public blockchain for our platform. With privacy being a focus of our ride pooling platform, there should be no possibility to trace individual user activity by monitoring the chain activities, even if it is public. Therefore we prioritise the increased decentralisation of public chains. Using a public chain allows us to utilise generic public nodes to handle smart contracts. Thereby we do not need to build a private network of independent node providers to build a private blockchain. Other research papers also prove the feasibility of decentralised ride pooling platforms on public chains [MAA22] [JSD+21] [BMS+20]

Bathen et al. introduce RiderS [BFJ20]. Central to RiderS is the emphasis on user privacy, achieved through a privacy-first biometric technology. Instead of traditional passwords, users become their own unique identifiers, ensuring genuine system interactions. To fortify this ecosystem, blockchain technology is employed, offering benefits like decentralisation and auditability. Each participant, whether a rider or an autonomous vehicle, accesses the system via a ”Wallet”. This software client manages credentials, facilitates transactions, and serves as the primary gateway into the blockchain. Monetary exchanges within this ecosystem utilise a stable coin named ”Mobi”, anchored to various cryptocurrencies and fiat currencies. This system is very useful and should be adapted by our platform. By introducing a Crypto Exchange to the platform, we allow the users to pay with a variety of different currencies, including fiat currencies, while still utilising the advantages of crypto currencies in our platform. A standout feature is the emphasis on privacy. Users can generate

single-use addresses, ensuring anonymity for each ride. This also should be adapted by our platform. Even though the wallet owner is anonymous on the chain, it prevents wallet tracking over a long period of time, which could lead to the exposure of the wallet holder.

Zhang et al. present a novel ride-hailing approach using blockchain and attribute encryption. [ZZHH22] The system includes a decentralised Blockchain-Based Ride-Hailing Mode. This mode has roles such as the Passenger, who generates encrypted ride details; the Driver, who decrypts and decides on ride acceptance; the Location Prover, verifying the driver's location; and the Authentication Center, distributing keys and authenticating identities. Thereby, the paper introduces a number of concepts that help us create our privacy-preserving ride-pooling platform. First of all, the concept of creating a shared secret between the customer and ride provider should be used to share sensitive information on-chain, like exact coordinates. With the Authentication Center, the paper also introduces an off-chain authentication service, which further promotes the concept of an off-chain authority that can verify wallets to handle on-chain interactions with the ride-pooling platform.

Mahmoud et al. propose a decentralised ride-sharing system to address challenges in centralised services, such as security concerns and single points of failure. [MAA22] The solution integrates blockchain with the Interplanetary File System (IPFS). Instead of storing all ride-sharing data on the blockchain, the system moves this data to IPFS and only retains a compact hash on the blockchain. This approach reduces data storage on the blockchain, leading to faster processing and lower costs. The system uses smart contracts on the Ethereum platform for management, and experimental results highlight its scalability and efficiency. This concept should be utilised if the prototype implementation or future iterations of the platforms should struggle with managing the amounts of data necessary to handle rides, resulting in high gas prices or slow blockchain performance.

Joseph et al. [JSD+21] introduce a sophisticated ride-matching system, utilising geolocation tools to pair riders with nearby drivers. While our platform is utilising an auction-based approach to match customers with ride providers, this paper showcases the advantages of an off-chain matching approach to handle the complex matching with an on-chain ride handling that tracks the actual ride.

Baza et al. introduce a decentralised system using a public blockchain, eliminating the central third-party vulnerabilities. [BMS+20] This system ensures location and time privacy by employing spatial and temporal cloaking techniques, allowing riders and drivers to share generalised locations and time intervals instead of exact details. This approach should also be utilised with our platform. With a location matching based on approximated data, we can ensure that the customer only needs to share their exact location with the ride provider that will fulfill the ride request. BlockWheels participants also use changing pseudonyms for each trip, ensuring untraceability. With BlockWheels also promoting this concept, it shows that this approach to ensuring untraceability is a best practice in regard to on-chain user activities. The entire scheme has been practically implemented and tested on the Ethereum platform, showcasing its feasibility and effectiveness in real-world scenarios.

The authors' Pal and Ruj introduce a decentralized ride-sharing solution using blockchain [PR19] . BlockV ensures fairness in ride-sharing in two main ways: Payment Fairness: It allows any network peer to compute the ride cost based on path details. Ride Fairness: In case of disputes, the system collaborates with Road Side Units (RSUs) to determine and penalize any malicious activity by drivers or riders. The BlockV system involves four key participants: the DRIVER, RIDER, BlockV, and RSUs. The process starts with riders selecting a route and fare from a decentralised database.

Once chosen, they confirm the ride and lock in the fare. At the ride's end, riders can either complete the ride, release funds, or raise complaints if unsatisfied. The system then verifies complaints using RSUs and takes appropriate action. With the RSUs, BlockV provides a solution to the problem of how to manage edge cases like customer complaints. While this concept relies on the existence of RSUs and mainly focuses on the handling of false routes taken by the ride provider, it showcases the importance of robust edge case handling.

Badr et al. propose a method of dividing the ride-sharing coverage area into small cells using overlapping grids [BBA+21]. This ensures that customers and ride providers are matched with location accuracy, as they report their locations by cell numbers. When their exact locations coincide within a common cell across any grid, a match is made. While this approach does not utilise the planned auction system proposed by our platform, it promotes a grid-based approach that can help to ensure that potential matching services can be bound to specific areas. By assigning matching services to single tiles in a grid, we can ensure that each customer can find their local matching service and that no matching service can collect data for areas that are too large.

Sefraoui et al. address privacy concerns in their paper by also utilising spatial cloaking and an off-chain matching service [VL21]. When a passenger requests a ride, an off-blockchain algorithm matches them with suitable drivers based on this cloaked data. To foster a sense of trust, both parties, the ride provider and the customer, post a deposit fee through a smart contract. This deposit acts as a commitment, and if either party defaults, the other is automatically compensated. This flow very much aligns with our vision of the interaction flow of our decentralised ride-pooling platform. The main advantage of this approach is that it allows for more complex matching algorithms without dramatically increasing gas fees while still utilising the advantages of blockchain by tracking the actual ride and related payments on-chain.

Sánchez et al. focus on preserving user privacy [SMD16]. In practice, this means that only when a driver's and passenger's trips align will they be privy to each other's identity, desired trip details, and reputation. This selective disclosure ensures that personal data remains confidential. This also aligns with the research objectives of our decentralised ride-pooling platform and needs to be considered in the final design. Addressing privacy alone is not enough; trust is equally important. The authors tackle this by weaving in a decentralised reputation management mechanism. Post a shared ride, both drivers and passengers have the liberty to rate each other. This allows peers to gauge the aggregated reputation of others based on historical ratings in a manner that's both transparent and trustworthy. This is a common best practice even with centralised ride-sharing platforms. For our decentralised platform, the rating should also be managed on-chain, as it profits from the tamper-proof nature of blockchain.

2.4.3 Conclusion

The detailed literature review shows that there are many different approaches to how a decentralised ride-pooling platform should be designed, with different authors focusing on different aspects of the platform. While there are many common best practices in regard to safety and user privacy, there is also no uniform approach to designing the different components of the platform. While some papers suggest handling all interactions with the platform on-chain, others suggest taking some elements off-chain to allow for more complex flows. Therefore, we can not rely on simply combining the platforms from the research papers into a single, feature-complete platform.

Therefore, to create a feature complete ride pooling platform, it will be necessary to make design decisions that will contradict the suggested approaches of some papers to embrace design decisions made by other papers. These decisions will be made based on our research objectives, which state that the maximisation of privacy, security and transparency is the underlying goal of our platform.

3 Proposed Solution

Having delved into the details of the current research landscape surrounding decentralised ride-pooling platforms, we can now construct our own platform based on the findings of the scientific literature. The goal is not to replicate existing platforms but to set a benchmark that encapsulates best practices from various research papers while also proposing improvements to current methodologies.

We want to look at the platform holistically and incorporate all the attributes in our design that are necessary to operate the platform, including Blockchain utilisation, transparency, user and provider interaction protocols, payments and service fees, privacy and anonymity, security and resilience, trust mechanisms, off-chain edge cases, and a prototypical realisation.

Therefore, in this chapter, we provide an overview of the conceptual design of the decentralised platform, discuss the inner workings of each component that is part of the platform in detail and showcase the data privacy and trust mechanisms of the platform. Through these explanations, we aim to provide a comprehensive blueprint for a decentralised ride-pooling platform that meets and exceeds the expectations set by the current academic and industry standards.

For convenience, the proposed platform will, from now on, be called GETACAR. We chose the name for its ability to describe the core offering of the platform, to get-a-car ride, while also being short, recognisable and easy to remember.

(The name also sounds phonetically similar to the title of the 1997 science fiction movie Gattaca, starring Ethan Hawke, Uma Thurman, and Jude Law, which the author of this paper immensely enjoyed.)

3.1 Conceptual Design of the Decentralised Platform

For ride-pooling platforms, the user experience for customers and ride providers is most important. From a customer perspective, the GETACAR platform's conceptual design is intentionally straightforward. GETACAR aligns its user flow with established centralised solutions such as Uber Pool and Lyft [Ube23] [Lyf23].

The reason behind this design decision is evident. Platforms like Uber and Lyft have invested significant resources to optimise user flow. Over the years, they have collected valuable insights and established best practices that have proven effective. It is counterproductive to reinvent the wheel when these robust ride flow models already exist. Instead, by basing the ride flow on these best practices, GETACAR aims to provide an experience that is not only familiar to users but also efficient and intuitive. One of the primary objectives of GETACAR is to provide an offer that rivals (if not surpasses) current centralised solutions. GETACAR aims to get users to transition from centralised platforms to a decentralised counterpart by emulating the ride flow of these established

platforms. To further promote the transition to GETACAR, the platform offers increased privacy and competitive pricing. Competitive pricing is achieved by offering reduced platform fees, which translates to better end-user prices.

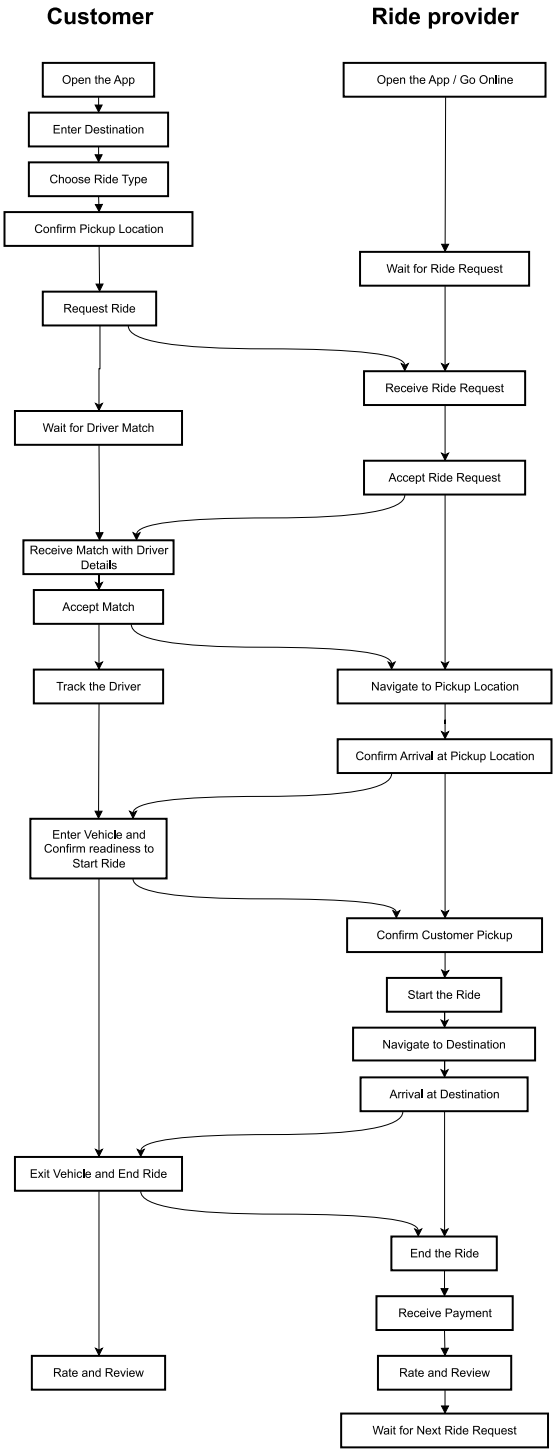

Figure 3.1: Ride Booking Flow

As shown in diagram 3.1, a generic customer-ride provider flow looks as follows: The customer journey begins by opening the app. This is followed by entering their destination. They then choose the type of ride they prefer by adjusting their ride settings and confirming their pickup location. Once these steps are completed, they request a ride and wait for a driver match. After receiving a match, the customer can track the driver's location in real-time. The customer enters the vehicle when the driver arrives and confirms their readiness to start the ride. This is a crucial step with autonomous vehicles; it lets customers signal their readiness. At the destination, the customer exits the car, ends the ride, and has the opportunity to rate and review their experience.

The process starts on the ride provider's side by opening the app if the ride provider is human. Autonomous vehicles will just need to connect to the platform and set their status to online. The ride provider then waits for a ride request. After receiving a request, they can choose to accept it. If accepted, the ride provider navigates to the pickup location. Upon arrival, they confirm to the customer that they arrived at the pickup location. The provider begins the drive to the dropoff location after the customer confirms they are ready to start the ride. Once the ride is concluded, the driver receives their payment and, like the customer, has the chance to rate the experience. After this, the driver waits for the subsequent ride request, completing the cycle.

Offering rides and interacting with the platform as a ride provider must also be designed to be as straightforward and user-friendly as possible. To compete with existing centralised platforms, which already employ huge fleets of ride providers, GETACAR needs to ensure a massive influx of ride providers. Therefore, it is essential to ensure that it is easy to connect autonomous vehicles to the platform and allow human ride providers to interact with it.

The decision to primarily focus on the interaction between a single ride provider and a single customer, without putting extra emphasis on the ride pooling aspect of the platform, was done so on purpose. For our vision of a decentralised ride-pooling platform, the difference between a classic taxi service and a ride-pooling service is minimal. From a customer point of view, the only difference between a taxi service and our ride pooling service is that the customer can select how many passengers are allowed to ride with them, what their minimum rating is supposed to be and how much time they are willing to take into account for detours to pick up other passengers. While a customer does not need to share their ride with other customers, they are incentivised to do so, as it results in lower overall ride cost and higher chances of finding matches.

On the other hand, this operation model creates a very competitive market for ride providers. Ride providers can maximise their sales volume by taking on multiple customers at once for a ride pool. But with an increased number of customers joining a ride, the chances of delays also increase, which can result in bad customer reviews. This constant decision-making about which customer to offer a ride to and which not allows for various operation strategies by the ride providers and promotes calculated decision-making and risk-taking.

3.1.1 Organisational Overview

While our research focuses on creating the technical architecture and design for our feature-complete decentralised ride-pooling platform, it is necessary to consider the organisational structure behind the platform. For the platform to grow, it is most important to have some form of supervising entity capable of developing updates for the technical components, representing and promoting the platform to external parties and verifying the trustworthiness of some components. When

implementing this kind of supervising entity, it is necessary to ensure that its centralised nature does not revert to the decentralised nature of the ride-pooling platform. Luckily, many examples from open-source software projects successfully showcase how to enable such an organisational structure. One example of this is the Open Source Initiative (OSI). OSI is a California-based public benefit corporation that has been educating about and advocating for the benefits of open source since 1998 [Ini23]. Therefore, we will assume, for the design of the technical components, that a so-called GETACAR Foundation exists that can handle organisational tasks necessary for creating a feature-complete ride pooling platform. To fund the foundation code is implemented into the smart contracts that transfer 10% of the overall ride cost of a successfully completed ride to a wallet managed by the GETACAR Foundation. The exact structure of this organisation should be part of future research in this field.

3.1.2 Component Overview

As previously discussed, ensuring that the user experience remains intuitive and seamless for all parties is important. To achieve this, the GETACAR platform is built upon several interconnected, decentralised components, each serving a distinct purpose to allow for an intuitive user flow that does not comprise security, transparency and privacy. The components can be seen in figure 3.2.

In the following, we will provide an overview of these components:

1. **Customer Frontend**:
    - **Purpose**: This is the primary interface for customers to interact with the platform.
    - **Features**:
        - Allows customers to request rides.
        - Enables customers to rate ride-providers and fellow passengers.
        - Provides settings for customers to specify preferences, such as the minimum acceptable rating for ride providers.
    - **Mode of decentralisation**: Self-hosted
2. **Ride Provider Frontend**:
    - **Purpose**: This interface is tailored for ride providers to manage their services.
    - **Features**:
        - Enables ride providers to view and bid on open ride requests.
        - Allows ride providers to rate passengers.
        - Provides settings for ride providers to specify preferences, like the minimum acceptable rating of passengers.
    - **Mode of decentralisation**: Self-hosted
3. **Authentication Service**:
    - **Purpose**: To ensure the security and privacy of user data.

- **Features**:
  - Manages customer and ride provider accounts and their associated ratings.
  - Ensures that customer and ride provider pseudonyms are kept separate from their real identities, providing an added layer of privacy.
- **Mode of decentralisation**: Self-hosted

4. **Matching Service**:
   - **Purpose**: To optimise the pairing of customers with ride providers.
   - **Features**:
     - Facilitates an auction mechanism where customers post ride requests visible to all ride providers in the vicinity.
     - Ride providers can anonymously bid on these requests, ensuring competitive pricing and optimal matching.
   - **Mode of decentralisation**: Self-hosted / Blockchain based
5. **Ride Contract Service**:
   - **Purpose**: To manage the intricacies of the ride and payment process.
   - **Features**:
     - Manages each phase of the ride process, from initiation to completion.
     - Handles the ride and payment process, ensuring secure and auditable transactions between customers and ride providers.
   - **Mode of decentralisation**: Blockchain based
6. **Crypto Exchange**:
   - **Purpose**: To bridge the gap between traditional fiat currency and the cryptocurrency used within the platform.
   - **Features**:
     - Allows customers and ride providers to transact in fiat currency for their real-world needs.
     - Facilitates the conversion between fiat and cryptocurrency, ensuring that all platform transactions remain crypto-based for added security and transparency.
   - **Mode of decentralisation**: Blockchain based

Each of the components is decentralised, either in the sense that the component is designed so that an individual or a group is able to run and host it themself, to interact with the network, or the component runs decentralised on the blockchain. After providing a small overview of the important components of the GETACAR platform, we will use the following Sections of this Chapter to dive deeper into the concepts and architecture behind each component.

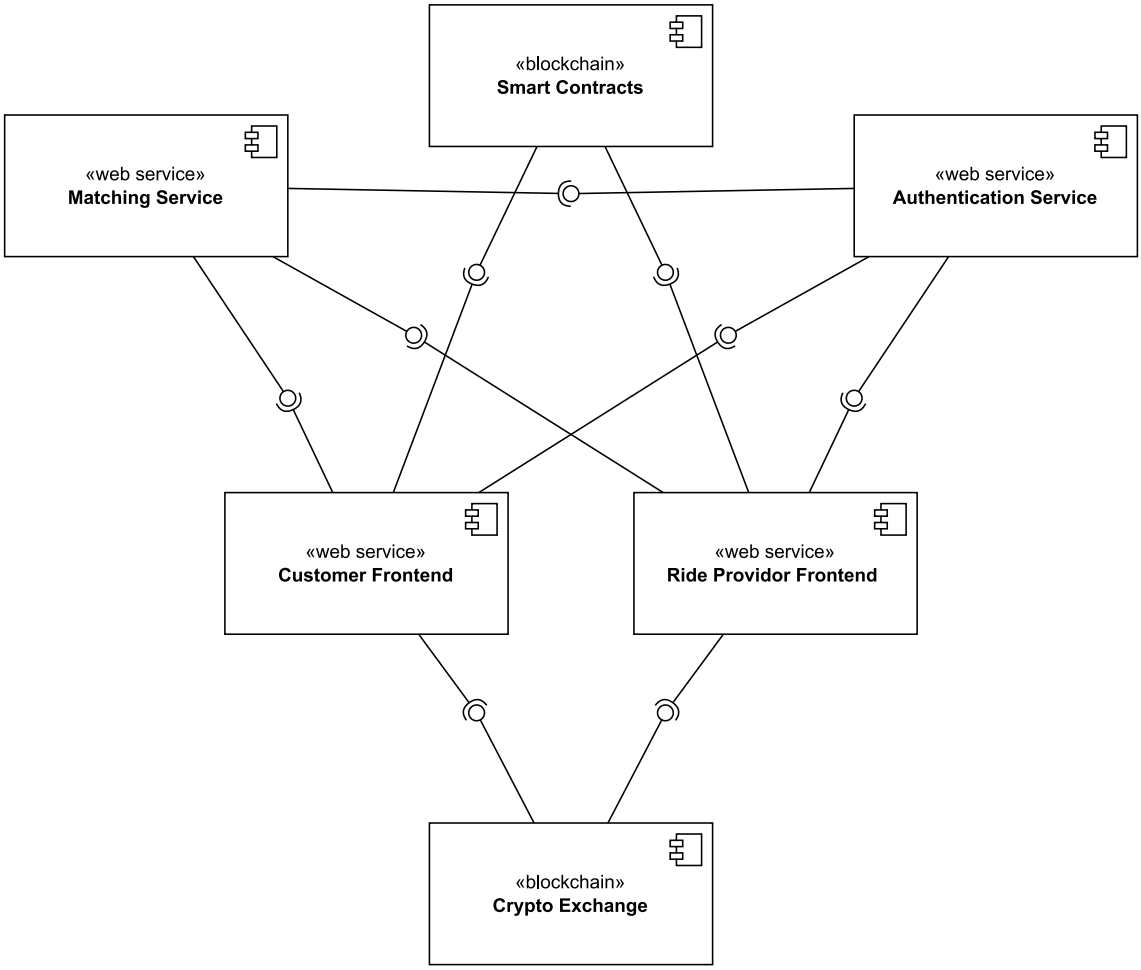

Figure 3.2: UML Component Diagram: GETACAR Platform

3.1.3 Customer Frontend

In the context of creating a decentralised ride-pooling platform, the user interface plays an important role in ensuring a seamless experience for both customers and ride providers. Therefore, the GETACAR frontend also draws inspiration from established platforms like Uber and Lyft. To build a comprehensive ride booking frontend, three distinct views are needed:

The primary point of interaction for customers is the booking view. This view lets customers easily input their desired pickup and dropoff locations through a search bar. Once the locations are set, this view previews the planned route on a map. This allows the customers to have a visual representation of their upcoming trip. Alongside the map, customers are presented with some additional details, such as the expected arrival time at the trip destination and the total distance of the ride. A button allows customers to commit to the route and request their ride. This initiates the matching service to find a fitting match for the customer.

Once a match is found, the frontend switches to the on-ride view, guiding the customer through the ride experience. The view first presents important information about the matched ride provider. This information includes the vehicle type, the ride provider rating, the number of passengers

already in the vehicle, the expected pickup time and the maximum price that the ride will cost. Two distinct buttons allow customers to either accept the ride offer or decline it. The customer has a fixed amount of time to accept the offer before it gets automatically declined.

Accepting the offer results in the customer depositing the maximum ride cost. After that, the view transitions the customer to the next phase of the on-ride flow, where the customer receives real-time status updates about the ride provider's current status. This includes "Vehicle is on its way", "Arrived at Pickup Location" and "Arrived at Dropoff Location". Once inside the vehicle, customers can confirm they're ready to start the journey and, upon reaching the destination, confirm the ride's successful completion. After completing the ride, the customer will get paid back any additional money they deposited that exceeded the actual ride cost. At any point between ride confirmation and completion, the customer has the option to abort the ride. The complete ride flow from the customer's point of view can be seen in figure 3.3. In that case, the money automatically gets deposited to the ride provider. If the reason for the aborted ride lies with the ride provider, the customer can contact the GETACAR foundation to initiate a complaint. Aborting rides because of the misconduct of ride providers is considered an edge case, especially when considering that most ride providers are expected to be autonomous vehicles. After the ride, customers can rate the ride provider and potential passengers, which is a crucial part of the platform's trust mechanisms.

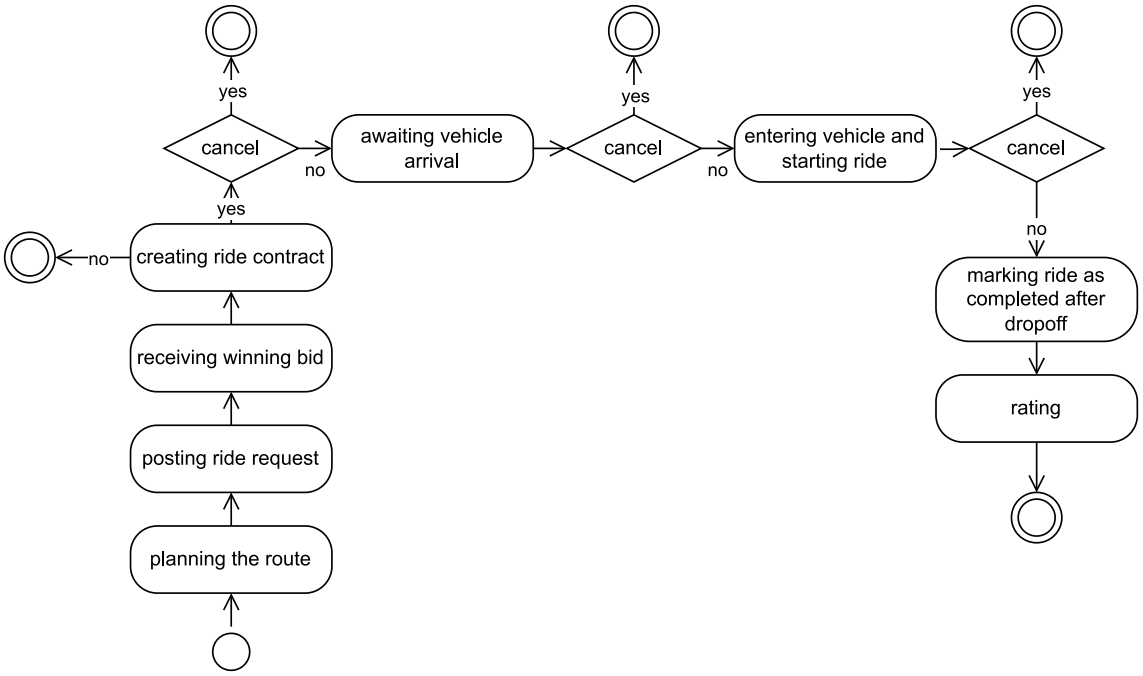

Figure 3.3: UML Activity Diagram: Ride Booking Flow

A settings view allows customers to manage their accounts and customise their ride experience. A customer can look up details about their account, including their rating. They can also adjust their ride preferences by specifying a minimum rating for ride providers. This ensures they are only matched with ride providers that meet their standards. Similarly, customers can adjust the minimum rating required for passengers to share a ride with them. This setting allows the customer to ensure a comfortable ride environment. This section also allows customers to manage other account settings. This includes payment methods and other privacy preferences. In conclusion,

the frontend design consisting of the booking view, the on-ride view and the settings view aims to provide an intuitive experience for customers through a focus on customer needs. The design of the GETACAR frontend will contribute to the platform's success and customer adoption.

To ensure transparency, the code of the frontend is open source. This allows everyone who is interested in it to take a look at the structure of this component in detail. This also allows customers to make personal adjustments to the application and to compile the frontend themselves.

3.1.4 Ride Provider Frontend

The design and functionality of the Ride Provider Frontend largely depend on the nature of the ride provider, be it a human pilot or an autonomous vehicle. To ensure a wide adoption by ride providers, the GETACAR Platform aims to cater to both types of providers.

For human ride providers, GETACAR provides a frontend similar to the customer frontend. This interface presents a view that allows ride providers to scroll through open ride requests. This view allows them to check ride details and place bids on ride requests. When an auction for a ride request results in the provider winning the ride opportunity and the customer confirming the ride, the frontend informs the ride provider that they have indeed won the auction. To confirm the ride and to receive the exact pickup and dropoff location of the customer, the ride provider needs to deposit 10% of the customer deposit themselves to ensure that they will actually carry out the ride. The ride provider can claim their deposit back once the ride is successfully completed.

After this confirmation, a second view becomes visible that allows the ride provider to manage the actual ride process. This flow mirrors the customer's journey. The provider confirms their commitment to the customer to take on the ride request and confirms their drive to the pickup location. When reaching the pickup point, the ride provider informs the customer of their arrival. The customer can then board the vehicle and initiate the ride. The ride provider then starts the travel to the dropoff location. After reaching the destination, the ride provider confirms the arrival via the frontend. This concludes the ride from their perspective. Once the customer leaves the vehicle and confirms the ride's end, the provider is able to claim the actual ride cost. This cost is deducted from the maximum ride cost deposit made by the customer at the ride's start. Just like the customer, the provider retains the option to abort the ride at any point between the ride's confirmation and the confirmation of arrival at the pickup location. The complete ride flow is shown in figure 3.7

Like the customer frontend, the Ride Provider frontend also includes a view offering general information and settings, mirroring the customer frontend. It is worth noting that all the functionalities are present, even if the ride provider is an autonomous vehicle. The distinguishing factor here is the absence of a visual user interface for autonomous vehicles. Instead, the computer that controls the autonomous vehicle can interact directly with the endpoints of the platform.

3.1.5 Authentication Service

The linchpin of security and user anonymity on the platform is the authentication service. Any user, be it a customer, ride provider, or a hoster of a matching service, must first register with an authentication service to interact with the platform. This service is the only component that manages user data such as name, age, address, and the associated rating. It also oversees the registration of

new autonomous vehicles and human ride providers, storing details like the number plate, VIN number, the responsible entity (company or individual) for the vehicle, and the vehicle's associated rating.

Once registered, users can use the authentication service to receive login tokens and pseudonyms. This also includes verifying crypto wallets that the user themself generates. The pseudonyms and credentials enable users to validate their permissions to engage with other platform components without sharing personal information, as seen in figure 3.4. For on-chain interactions, users can use the frequently changing wallets linked to their authentication service account. This approach not only ensures user anonymity during on-chain engagements but also prevents external entities from monitoring the chain and tracking user activities.

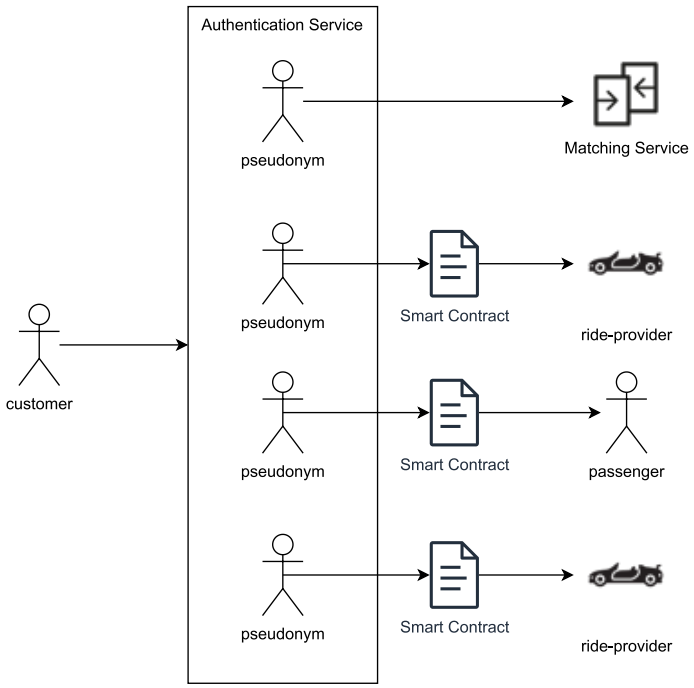

Figure 3.4: Customer Pseudonyms trough Multiple Components

The architecture also facilitates rating tracking by the authentication service. For example, a customer utilises five different crypto wallets across five rides and receives five ratings from ride providers. Monitoring of the blockchain only reveals five separate wallets, each with a single rating, between a pool of other wallets on the platform with individual ratings. The authentication service, however, is able to connect all five wallets to the customer, as it knows all pseudonyms and wallets linked to the customer, enabling it to compute an aggregate rating. The authentication service also ensures that users can not register multiple times with the platform to escape a series of bad ratings, for example.

Beyond user authentication and rating computation, the service also validates rating information about other users. For example, a ride provider can verify a user's claimed rating by forwarding the rating and the user's pseudonym to the matching service.

In edge cases, the authentication service can unveil a user's identity, such as when a ride provider seeks to make an insurance claim to a customer due to intentional vehicle damage by a customer or when a passenger wishes to report harassment to law enforcement. While revealing a user's identity should be rare, it's an indispensable feature to ensure platform security and safety.

Given the immense power held by the authentication service, it is important to prevent it from morphing into a centralised control point. Therefore, the design allows anyone to host their authentication service and request it to be verified by the GETACAR foundation. First-time platform registrants can also choose their preferred authentication service from the list of verified services. All verified authentication services communicate with each other, ensuring that users can not secretly create identities across multiple services. This inter-service exchange is executed without revealing the users managed by each service to one another.

3.1.6 Matching Service

The primary responsibility of the matching service is to match ride requests from customers with suitable ride providers. Traditional centralised services, such as Uber, employ algorithmic matching systems [Ube23]. These algorithms consider various factors like proximity, availability, and customer preferences to quickly assign a driver to a customer's request. While efficient, this centralised approach often lacks transparency, and the decision-making process is entirely controlled by the platform, potentially leading to biases or unfair advantages for certain drivers or riders.

Recognising these challenges, GETACAR has adopted an auction-based ride-matching system. This approach offers several advantages. Firstly, it introduces a competitive environment where ride providers can bid for rides, ensuring that customers get the most cost-effective offers. Secondly, it promotes transparency by making the selection method known to everyone. Lastly, it empowers ride providers by giving them the autonomy to bid on rides based on their preferences and profitability. The complete matching flow between the customer, ride provider and platform can be seen in figure 3.5.

Within this framework, GETACAR employs a Vickrey auction for its matching. The second-price Vickrey auction has distinct advantages. In this model, the highest bidder wins but pays the amount bid by the second-highest bidder. This encourages ride providers to bid their true valuation without the fear of overpaying. It promotes honest bidding, reduces the chances of strategic manipulation, and ensures that customers receive competitive prices while providers are fairly compensated. Building upon the generalised, anonymous second-price auction concept, the intricate auction flow within the matching service unfolds as follows:

A customer initiates the process by submitting a ride request to the matching service. This request encapsulates an approximation of both the pickup and drop-off locations, ensuring that the precise coordinates are exclusively shared between the customer and ride provider once a mutual commitment to the ride is established. Additionally, the request contains ride details such as the customer's rating, the minimum acceptable rating for the ride provider, the minimum rating for potential co-passengers, and the maximum passenger count.

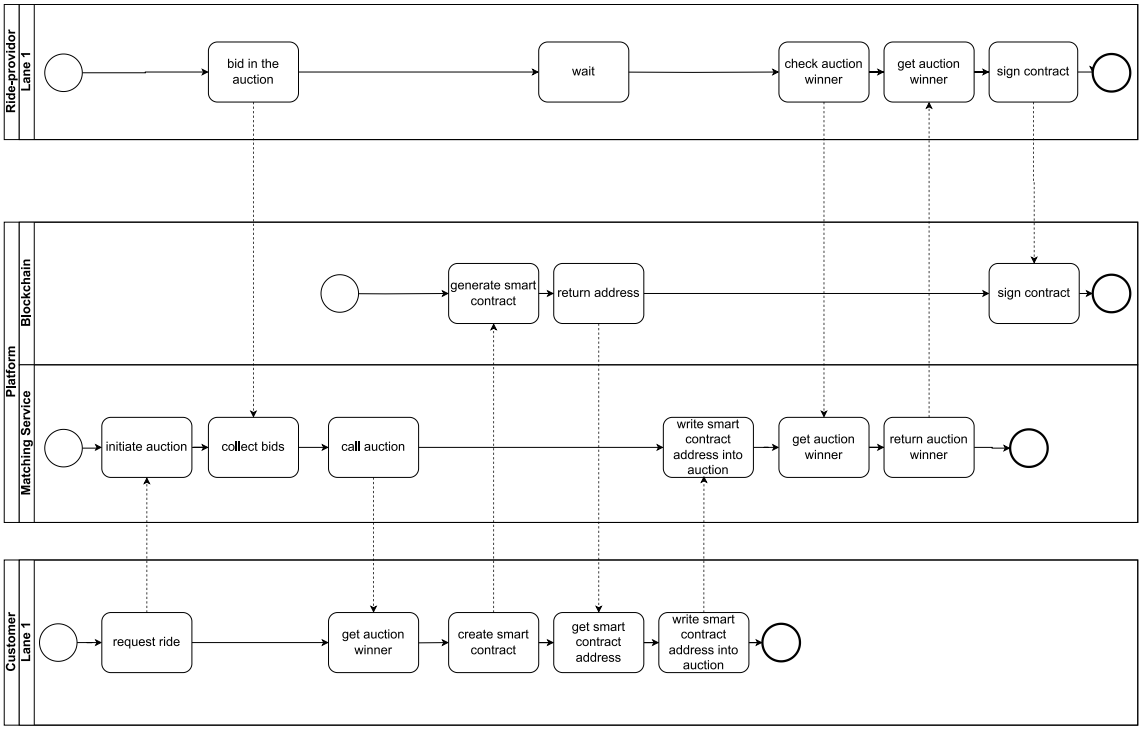

Figure 3.5: BPMN 2.0 Activity Diagram: Customer, Ride Provider Matching Service Flow

Upon receiving the ride request, the matching service marks it as an open auction, inviting ride providers to place their bids. This auction remains accessible for a predetermined duration, during which ride providers can submit their bids. These bids contain the proposed ride price and are accompanied by additional data like the provider's rating, anticipated arrival time and the vehicle model.

Following the auction's closure, the matching service identifies the winning bid and communicates the result to the customer. Subsequently, the customer is granted a fixed time window to review and accept the offer before it lapses.

Should the customer confirm the winning bid, they generate a ride contract containing the agreed-upon sum from the bid. The winning bidder is then notified of their successful bid and receives the ride contract's address, marking the commencement of their service obligation.

Lastly, the matching service will be used to establish a secure connection to share sensitive information on-chain. Therefore, the matching service initiates a Diffie-Hellman Key Exchange by sharing the necessary prime number, base and submitted customer and ride provider public keys between the two parties. Based on this information, the customer and ride provider can compute a shared secret for encrypted interactions on the blockchain.

This secure connection will allow customers and ride providers to share detailed location data and other bits of personal data via blockchain without exposing the information to the ledger.

Having explained the core concept of the matching service, it is important to also explain how decentralisation is achieved for this service and how we ensure that the matching service can not collect too much transaction data. One conceivable method to ensure decentralisation is to execute the matching process on-chain. Running the matching service on-chain offers transparency, immutability, and a trustless environment, ensuring that all transactions are verifiable and irreversible.

However, this on-chain approach is not without challenges. Even with the use of pseudonyms and location approximations, all ride requests and bids would be publicly accessible on the blockchain. This transparency, while advantageous in some contexts, could possibly expose patterns, preferences, and behaviours of users, potentially compromising their privacy and anonymity. Moreover, executing the intricate matching flow on the blockchain, especially at a large scale, demands significant computational resources. This would inevitably increase the platform's overall costs, thereby inflating ride prices and reducing the profits for ride providers.

To circumvent these challenges, the platform adopts an approach similar to the authentication service. Individuals or organisations can host their independent matching service, which, post-registration, undergoes verification by the GETACAR foundation.

To ensure an equal distribution of ride requests between the matching services and to prevent matching services from collecting excessive amounts of personal data, a grid-based distribution system is proposed. The grid-based system assigns each service-specific grid field as their jurisdiction zone.

To stop a single service from collecting long-term data within a grid, we mandate that each grid be managed by at least two distinct matching services. The GETACAR Foundation also has the possibility to rearrange the grid jurisdiction zones in time intervals for additional security.

For a customer, the process of finding the right matching service looks as follows: Every customer frontend contains a list of all verified matching services and their corresponding grid jurisdictions. The customer is then able to determine the matching services responsible for their current grid square. Customers can then provide the list of possible matching services to an on-chain service and receive the best matching service from the available options within their grid. This on-chain service guarantees an even distribution of requests within a grid square by monitoring the number of requests for each service and the amount of successfully completed rides through the matching service. This allows for a load balancing between the matching services so that no service can collect too much data. Furthermore, it allows for the creation of a rating for every matching service, derived by dividing the total number of requests processed by the service by the number of rides successfully completed through it. The complete flow can be seen in figure 3.6.

3.1.7 Ride Contract Service

Within the blockchain, various events and statuses related to the ride-pooling process are recorded. Initially, the ride provider accepts the ride, and a "Ride-provider accepted" status is written to the blockchain. Following this, the ride provider starts driving to the pickup location, and once they arrive, a "Ride-provider arrived at pickup location" status is documented. Meanwhile, the customer indicates they are ready to start the ride, prompting a "Customer ready to start ride" status to be written to the blockchain. The ride provider then starts the ride, leading to a "Ride-provider started ride" status getting written onto the blockchain. Upon the ride's conclusion, when the ride provider

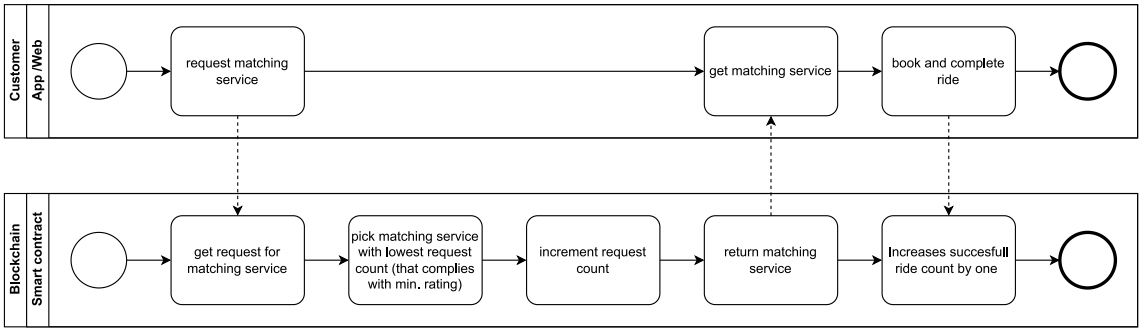

Figure 3.6: BPMN 2.0 Activity Diagram: Customer and Matching Contract Interaction

arrives at the drop-off location, a ”Ride-provider arrives at dropoff location” status is recorded. The customer then marks the ride as complete, resulting in a ”Customer marked ride complete” status on the blockchain. The complete flow can be seen in 3.7

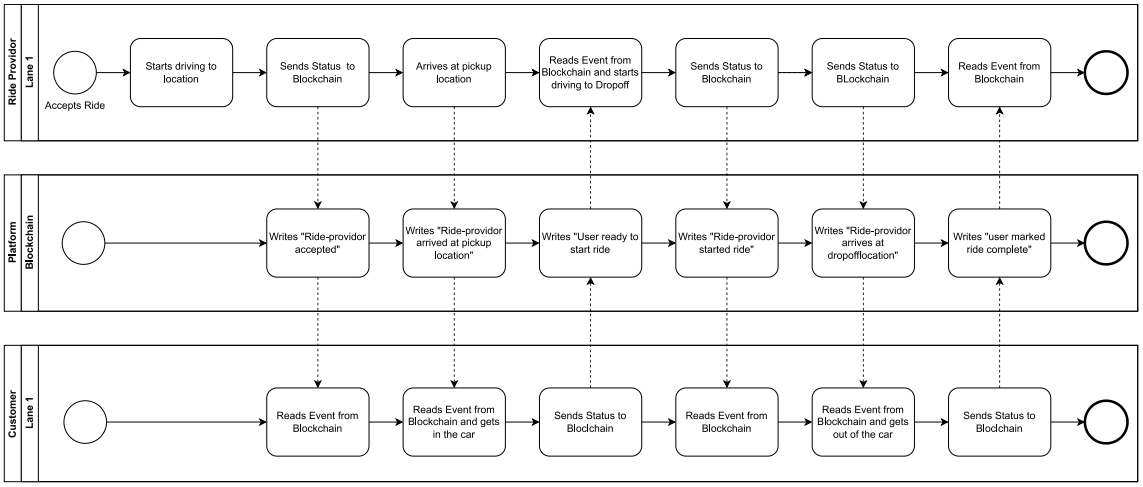

Figure 3.7: BPMN 2.0 Swimlane Diagram: Customer, Platform and Ride Provider

Each event that gets posted to the blockchain also contains a custom message that can be determined by the entity posting the event. These messages will be encrypted by the key that was exchanged between customer and ride provider via the Diffie-Hellman Key Exchange. These Messages can contain a number of different information depending on the status of the ride. A typical message sent by the ride provider as part of the initial “Ride-provider accepted” event would be a tracking link that allows the customer to track the vehicle. The contract also contains a number of additional functions in regard to trust and privacy that are described in detail in subsection 3.2.2 and subsection 3.2.3.

This decentralised ride-pooling platform ensures transparency and trustworthiness by recording every significant event on the blockchain, with both the customer and the ride provider actively interacting with it, ensuring a smooth and verifiable ride process.

3.1.8 Crypto Exchange

The Crypto Exchange plays an important role in bridging the gap between traditional fiat currencies and cryptocurrencies, ensuring that all transactions within the platform can utilise the advantages of cryptocurrencies. When a customer books a ride, the frontend instantiates a new wallet. To fund this wallet for the ride, the customer pays a crypto exchange, which then credits the wallet with the required amount of cryptocurrency. Post-ride, ride providers have the option to convert the cryptocurrency they have received into fiat currency via the crypto exchange.

This enables the platform to utilise the benefits of blockchain and cryptocurrency, such as transparency and security. Simultaneously, it offers the flexibility for customers and ride providers to transact in the more universally accepted fiat currency, thereby enhancing usability.

However, a significant privacy concern emerges when considering that most crypto exchanges mandate personal identification for buying and trading cryptocurrency. This could potentially establish a traceable link between real-world identities and wallet owners, undermining the platform's commitment to privacy. There are several solutions to solving this problem. One approach involves collaborating with an exchange that solely requires verification from a GETACAR authentication service to authorise trades, bypassing the need for personal identification. Alternatively, diversifying transactions across multiple crypto exchanges or employing coin mixers can obfuscate transaction trails. A coin mixer, or cryptocurrency tumbler, is a service that mixes potentially identifiable cryptocurrency funds with others, making it challenging to trace specific coins back to their original source. This ensures enhanced privacy and anonymity for users, making transactions less traceable on the blockchain.

Another proposition is for the authentication service to function themself as a crypto exchanges. This would further strengthen the platform's services, potentially offering a more streamlined user experience. The decision on which approach to flow depends on multiple external factors including the willingness of cooperation from crypto exchanges and the liquidity of the authentication services.

3.2 Privacy Measures and Trust Mechanism Design

GETACAR utilises several privacy and trust mechanisms to ensure the safety of all customers and ride providers. In the following section, we will describe these mechanisms in detail and showcase how personal data is shared across the platform.

3.2.1 Data Privacy from a Customer and Ride Provider Perspective

Now that we have explained the functions and concepts behind all the components relevant to the GETACAR ride pooling platform in detail, we can analyse how data privacy is affected by the platform's design.

To assess the data privacy of the platform based on its design, it is important to showcase how the personal data of customers and ride providers is shared between the GETACAR services.

Table 3.1 is a compilation of data points relevant to ride-pooling apps. The list is inspired by the data points revealed when exporting all personal data collected by Uber [1].

To facilitate clarity and ease of interpretation, each data point, in the context of each service, is represented using specific symbols:

- ✓: Symbolises that the service possesses access to the respective information.
- •: Denotes the necessity for the information to be available to this party.
- ×: Indicates the service's lack of access to the particular information.
- (×): Indicates that the service can access parts of the information.

The table provides a detailed breakdown of these data points about the various services within the platform.

Table 3.1: Customer Data Privacy Matrix

Data	Customer	Ride Provider	Matching Service	Crypto Exchange	Authentication Service	Publicly Available
Basic Personal Details:						
Full name	✓•	×	×	×	✓•	×
Gender	✓•	×	×	×	✓•	×
Date of birth	✓•	×	×	×	✓•	×
Contact Information:						
Email address	✓•	×	×	×	✓•	×
Phone number	✓•	×	×	×	✓•	×
Home address	✓•	×	×	×	✓•	×
Payment Information:						
Credit/debit card details	✓•	×	×	✓•	×	×
Bank account details	✓•	×	×	✓•	×	×
Payment history	✓•	×	×	(×)	✓	×
Billing address	✓•	×	×	✓•	×	×
Ride Details:						
Pickup and drop-off locations	✓•	✓•	(×)	×	×	×
Date and time of rides	✓•	✓•	(×)	×	✓	×
Ride preferences	✓•	✓•	(×)	×	×	×
Ride history	✓•	×	×	×	×	×
Location Data:						
Real-time location during a ride	✓•	✓•	×	×	×	×

Continued on next page

[1] https://help.uber.com/en-GB/riders/article/whats-in-your-data-download-?

Table 3.1 – continued from previous page

Data	Customer	Ride Provider	Matching Service	Crypto Exchange	Authentication Service	Publicly Available
Frequent locations	✓●	×	×	×	×	×
Route taken during the ride	✓●	✓●	×	×	×	×
Device Information:						
Device type	✓●	×	×	×	×	×
Operating system	✓●	×	×	×	×	×
App version	✓●	×	×	×	×	×
Device identifiers	✓●	×	×	×	×	×
Communication Data:						
In-app messages between driver and rider	✓●	✓●	×	×	×	×
Behavioral Data:						
App usage patterns	✓●	×	×	×	×	×
Click patterns within the app	✓●	×	×	×	×	×
Features frequently used	✓●	×	×	×	×	×
Safety and Security Data:						
Records of any incidents or disputes during rides	✓●	✓●	×	×	×	×
Ratings:						
Ratings provided about drivers	✓●	×	×	×	✓	×
Ratings received from drivers about the user	×	✓●	×	×	✓	×
Preferences and Settings:						
Language preference	✓●	×	×	×	×	×
Notification settings	✓●	×	×	×	×	×

The GETACAR platform has been designed to focus on user privacy. Through its architecture, the platform ensures that user data is managed with the utmost discretion, granting access only where necessary across its components. The table provided offers a detailed overview of this data-sharing.

When it comes to basic personal details such as the full name, gender, and date of birth, these are exclusively accessible by the customer and the authentication service. This design choice ensures that these personal identifiers remain shielded and are not exposed to other platform components. Similarly, contact details like the email address, phone number, and home address are secure, with access limited to the customer and the authentication service. This setup ensures that personal contact details remain undisclosed to ride providers or other services. As explained in subsection3.1.5, the only time the authentication service exposes these data points to third parties is in extreme edge cases, for example, when a ride provider seeks to lodge an insurance claim due to intentional vehicle damage by a customer or when a passenger wishes to report harassment to law enforcement.

In the realm of payment information, details like credit/debit card numbers and bank account specifics are only shared between the customer and the crypto exchange. This arrangement allows for crypto transactions without sharing personal payment information with other parties. In addition the payment history is accessible

only to the customer if they utilise several different crypto exchanges or other methods to hide their payment history, as explained in subsection 3.1.8. The only other entity that has access to the payment history is the used Authentication Service, as it knows about all wallets linked to the user.

Ride details are at the centre of the platform's operations. Information such as pickup and dropoff locations, the date and time of rides, and specific ride preferences are shared between the customer and the ride provider. While most of the ride details are shared only between the customer and ride provider, the date and time of rides are also visible by the authentication service because the service can look up on-chain activities of users because the service knows all wallet addresses connected to one customer. While the matching service only receives this information linked to a pseudonym, it is important to follow the grid-based matching approach to ensure that the service can not unveil identities through the long-term collection of data.

Additionally, location data, including real-time location during a ride and the route taken, are encrypted and shared only between the customer and the ride provider executing the ride, as decided in subsection3.1.6.

Device-related information, which encompasses details like the device type, operating system, app version, and unique device identifiers, remains confidential, with access restricted to the customer. As decided in subsection 3.1.3, this is ensured by providing the frontend as open-source code with no user data collection features.

Communication, especially in-app messages between the driver and the rider, is encrypted, ensuring that all communication remains confidential as described in subsection 3.1.6. Behavioural data, which includes insights into app usage patterns, click patterns within the app, and frequently accessed features, is kept private, with access limited exclusively to the customer. This is ensured by providing the frontend as open-source code with no user data collection features.

Ratings form an integral part of the platform's security mechanism. Customer Ratings about drivers are accessible to both the customer and the authentication service. On the other side, driver ratings about the customer are accessible to the ride provider and the authentication service. This dual-access system ensures transparency in the rating process while also ensuring user privacy, as described in subsection 3.2.2.

Looking at the ride provider data privacy, the table 3.1 can equally be applied to the ride providers. Ride providers have some additional information managed by the authentication service, like the vehicle's number plate, its VIN and, in the case of autonomous vehicles, the individual or company owner that has authorised the vehicle to be part of the GETACAR platform. Like the customer contract information, they are not shared with other services. The same goes for interactions with Matching Services and Crypto Exchanges.

The table and its accompanying explanation underscore the GETACAR platform's commitment to securing user data. Thanks to pseudonyms and changing wallets, there is no publicly available data that unveils a user's identity or allows for long-term tracing to make assumptions about the identity of a user. This is an excellent achievement for a decentralised, transparent, blockchain-based platform. Additionally, the data available to services that do not explicitly need access to this information is reduced to a minimum and could be further reduced by applying complex and computationally intensive cryptography like homomorphic encryption, which could potentially impact the user experience. Another option would be utilising layer 2 solutions that might allow us to tackle some potential platform shortcomings. Solutions like the Raiden Network for the Ethereum blockchain[2] can increase transaction times and impede transactions from being logged on to the public ledger, but they also impact decentralisation, and the introduction of another component that has access to sensitive user data can be counterproductive to achieving the research goals [NIR+23]. Assessing the best possible ways to increase data privacy further can be part of future research.

[2] https://raiden.network

3.2.2 Rating Trust Mechanisms

Rating systems are an important tool for ensuring trust and safety on ride-pooling platforms and are considered a best practice based on the scientific literature. When passengers and drivers are allowed to rate each other, it creates a feedback mechanism that makes both parties accountable. Providing customers with the possibility of viewing the ratings of other customers and ride providers allows them to make informed decisions about which ride providers to choose based on previous experiences of other customers. Additionally, a rating system can promote better service from ride providers. Equally, customers are encouraged to show better behaviour towards passengers and the ride provider. As it is the goal of this research paper to design and build the technical foundation of the GETACAR platform, we will assume that a generic five-star rating system with time-weighted ratings provides optimal results. The GETACAR platform's design ensures that several rating systems are supported. Determining the best possible rating system for the GETACAR platform should be part of future research.

As discussed in 3.1.3 the platform allows customers to rate their ride providers and passengers and allows ride providers to rate their customers. The ratings are posted onto the ride contract running on the Blockchain through a rating function. This allows for much flexibility regarding the rating system because the smart contract can easily be adjusted to change the rating system, for example, from a five-star rating to a ten-star rating. The process of how ride providers and customers rate each other is seen in figure 3.8.

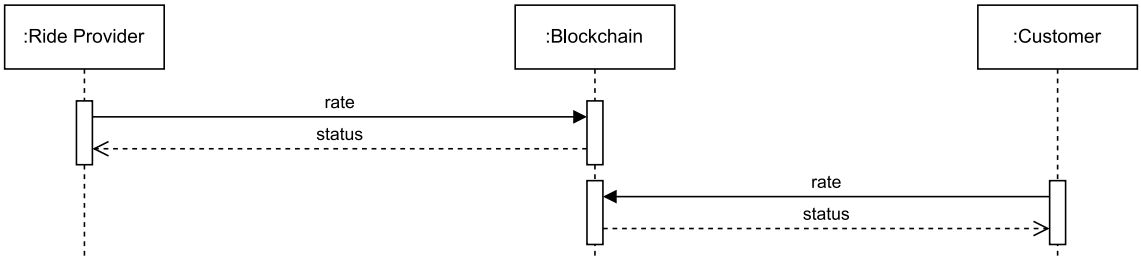

Figure 3.8: UML Sequence Diagram: Customer and Ride Provider rate each other on Blockchain

To allow customers to rate passengers, the ride provider shares a list of all passengers that are present during the ride. The shared list contains the passenger's pseudonym, the seating portion of the passenger and the start time of their ride. This information allows the customer to post ratings to each passenger without passengers needing to reveal their identities. This flow can be seen in figure 3.9.

The authentication service monitors all contracts. If a rating is posted on the Blockchain that affects a pseudonym that belongs to a user managed by an authentication service, this instance of the authentication service adds the rating to the user profile. Thereby, the authentication service has a list of all ratings that belong to a single user alongside metadata like the time the rating was posted. This allows the rating service to implement more complex rating systems that provide more accurate ratings.

To ensure that users do not lie about their ratings, the authentication services work as a verification service for the ratings they manage through a request that contains the non-verified rating and the pseudonym connected with it. The flow can be seen for customer and ride provider in figure 3.10 and 3.11. This by itself creates a small risk that users can find connected pseudonyms by analysing ratings posted on the Blockchain. To counter this, all ratings the authentication service provides (including the rating presented to a user as their own rating) are rounded to 0.3 steps (e.g. possible ratings are 5, 4.7, 4.3, 4, ...).

To put it in a nutshell, the combination of on-chain ratings combined with off-chain rating accumulation allows for more complex and accurate rating systems while still utilising the transparency and auditing abilities of a blockchain-based rating system, which ensures a secure and trustworthy environment on the platform.

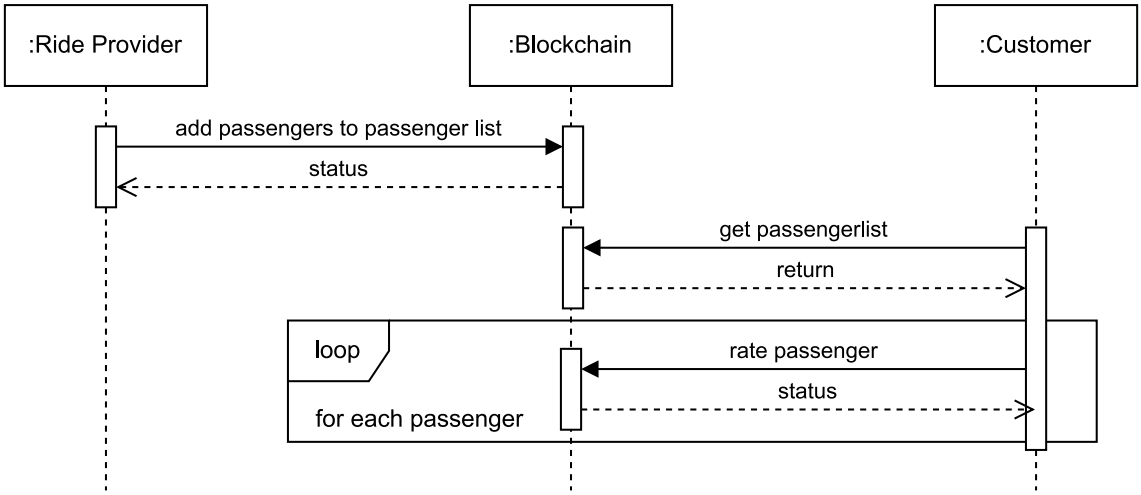

Figure 3.9: UML Sequence Diagram: Ride Provider adds Passengers to Ride Contract and Customer rates them

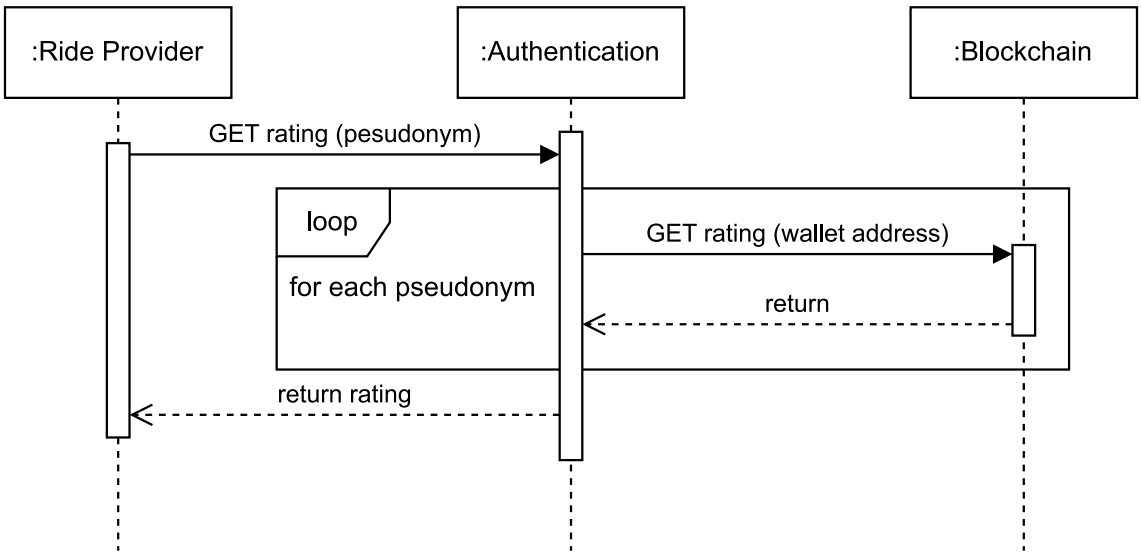

Figure 3.10: UML Sequence Diagram: Get Rating from Authentication Service as Ride Provider

Additionally, the completely blockchain-based rating system is in place for the matching services as described in 3.1.6, allowing customers to ensure that they are utilising a well-performing matching service without risking exposing personal information by utilising the same instance of the matching service too often.

3.2.3 Deposit Trust Mechanisms

While the rating system provides a solid foundation for ensuring trust in the platform, many research papers utilise money deposits by the customer and ride provider to ensure that the cost of the ride will be paid and to counter malicious activities on the platform. In the following, we will focus on how to implement such deposit functions into the platform with the help of smart contracts. It is not part of this research to determine what the exact amounts of deposited money should be to ensure trust between customer and ride provider. This should be part of future research.

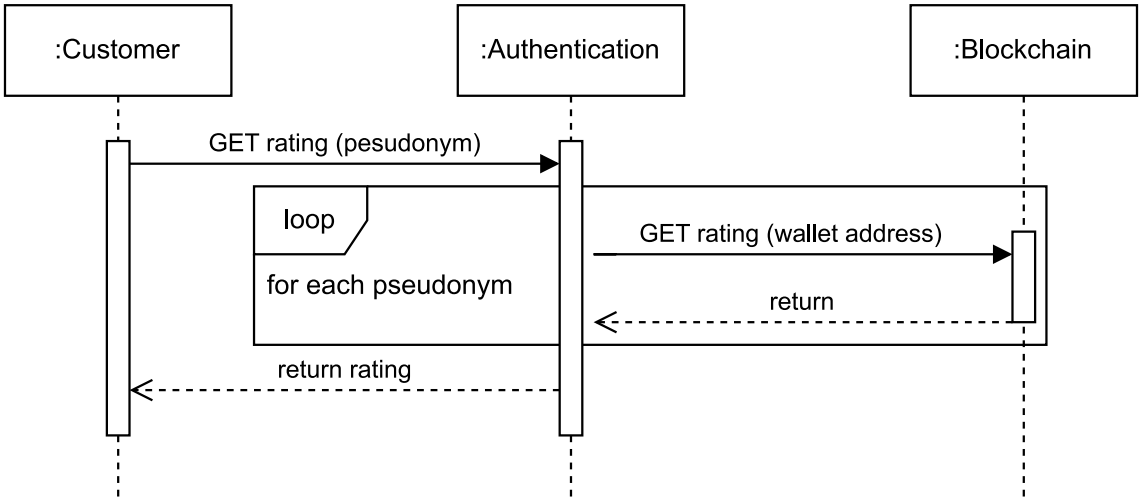

Figure 3.11: UML Sequence Diagram: Get Rating from Authentication Service as Customer

The implementation of the deposit-based trust mechanism in GETACAR looks as follows: After the customer reviews and accepts the winning bid from a ride provider, they create a ride contract on the blockchain as described in subsection 3.1.7. This contract is loaded with a deposit by the customer that is equal to the maximum ride cost set by the ride provider in their bid. This ensures that the ride provider has enough money to pay for the ride. It also ensures that the customer can not be overplayed by the ride provider as the maximum ride cost is limited by the deposit in the contract.

To ensure that the ride provider has honest intent on completing the ride, they have to deposit 10% of the maximum ride cost into the contract themselves to be able to be allowed to handle the ride. This deposit will be transferred back to the ride provider after completing the ride. After the ride is marked as completed by both the customer and the ride provider, the ride provider can now claim the amount of money from the ride contract that they determined as the actual ride price. The rest gets transferred back to the customer. By managing all the transactions through a smart contract, we can ensure that no trusted third party is needed as well, and no trust between the customer and ride provider is needed to ensure that all transactions are handled properly.

3.2.4 Conclusion

Based on the current state of the scientific literature and best practices from the industry, we designed the GETACAR ride-pooling platform. This chapter has shown how each of the components of the GETACAR platform works and how components interact with each other. The design was also assessed on how it handles data privacy, and we explained how trust mechanisms are implemented into the platform. Now that we have successfully designed the platform, it is important to show that the design successfully transports into reality.

4 Implementation of the Decentralised Platform

To validate the design of the GETACAR Platform, it is important to showcase that an actual implementation of the services is feasible. Therefore, in the following chapter, we showcase how each component is built. The goal of this implementation is not to create a ready-to-release platform but to prove that the core features and functions of the platform work as previously described. Because of that, the aim is to build this prototype with as many standardised and commonly utilised technologies as possible.

The goal is to present a prototype that can be used as a blueprint to build and release a marked-ready decentralised ride-pooling platform. Therefore, it should be easy to adapt the design of the GETACAR platform with a variety of different underlying tech stacks.

The prototype we describe on the following pages does cover the smart contracts that enable the ride flow, the matching service and both the frontend for the customer and the interface for the ride provider. All components interact with each other and provide a complete ride experience. Not part of the prototype is the authentication service. As described in the introduction of this paper, the realisation of a decentralised authentication service is not part of the scope of this research. Additionally, the prototype does not fully implement the Crypto Exchange component, as it would require corporations with multiple companies to provide crypto exchange services. Therefore, this prototype utilises a crypto wallet with an integrated crypto exchange to allow for the manual stimulation of the buying and selling cryptocurrency on the GETACAR platform.

4.1 Smart Contracts

Smart contracts make up the backbone of the GETACAR ride-pooling platform. They allow for the secure and transparent ride ordering flow that is the standout feature of the platform. Therefore, we will start with the construction of these smart contracts for the prototype.

Before writing the contracts, it is necessary to decide on a programming language. This decision is crucial because it will also affect the compatibility of the smart contracts with the available blockchain platforms.

Looking at the smart contract programming languages used by the research papers and the adaption of smart contract programming languages by blockchains, the decision is clear: Solitidy[1] is a broadly adapted smart contract programming language that is not only utilised by the Ethereum blockchain[2] but also other popular blockchains like the BNB Chain[3], Tron[4] and Avalanche[5]. Besides being widely adopted it is also utilised in a number of papers from the identified scientific literature to build decentralised ride-pooling platforms [BLM+21] [MAA22] [BMS+20].

[1] https://soliditylang.org
[2] https://ethereum.org/en/
[3] https://www.bnbchain.org/en/smartChain
[4] https://tron.network
[5] https://www.avax.network

Solidity is a high-level programming language originally tailored for the Ethereum blockchain's smart contracts. Influenced by JavaScript, Python, and C++, its syntax allows developers to craft self-executing contracts where terms are coded directly. These contracts are compiled to bytecode for the Ethereum Virtual Machine (EVM). Given blockchain's immutable nature and financial implications, Solidity emphasises security and exception handling [Sol23].

Selecting this smart contract programming language for the development of the GETACAR prototype ensures easy adaption by future developers. Additionally, the high adaption rate of Solidity ensures a future-proof reference implantation in the fast-evolving crypto landscape, compared to languages that are proprietorially used by smaller blockchains. For the creation of the prototype, the smart contracts are deployed on an Ethereum Blockchain Node and utilise ETH as their underlying currency for Gas Fees and payments. The complete smart contracts described in the following section can be found in the appendix of this paper.

4.1.1 Contract Factory

There are two design approaches to utilise a smart contract to manage all critical ride events. Firstly, it is possible to create a single, smart contract that can be used by all customers and ride providers to log their rides. The advantage of this approach is lower gas fees because no new contract is generated for every trip. The downside of this approach is that it drastically increases the complexity of the contract to ensure that all trips stay separated inside the contract. It also increases the impact of security loopholes in the contract because it could possibly allow users to influence the rides of other users.

The second approach would utilise the concept of a contract factory. A contract factory allows the generation of smart contracts based on a predefined template through a second smart contract, the so-called contract factory. The main disadvantage of a contract factory is increased gas prices because the deployment of a new smart contract is generally more expensive than the interaction with an existing one. But the contract factory approach also provides a number of upsides. It allows for each ride contract to be capsuled into its own smart contract, which improves security and decreases the complexity of the ride contract itself.

Because data privacy and security are most important to the design of the GETACAR platform, it is decided to follow the contract factory approach. The contract factory that is designed to generate ride contracts looks as follows:

At the centre of the contract factory exists the `createContract()` function that creates a new ride contract that takes the amount of ETH (that represents the maximum ride cost) as a deposit, as seen in listing 4.1. The deposit holder will be the newly created ride contract, which will only return the deposit if the right circumstances are met. `createContract()` also executes some additional code that helps authentication services track newly created contracts. Each contact gets assigned a contract number determined by the contract counter and is mapped to a timestamp that represents its creation date. Additionally, the contract gets registered to the Matching Smart Contract. The interaction between the Ride Contract and other contracts can be seen in figure 4.1. The reason for the interactions with the Matching Contract will be explained at a later point.

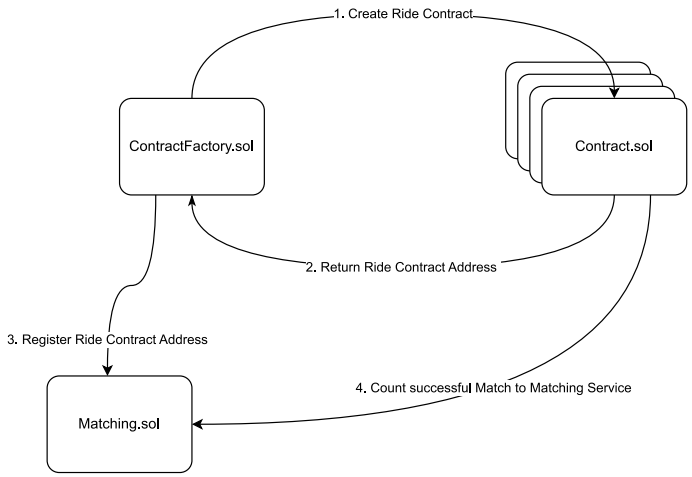

Figure 4.1: Interactions between Smart Contract

Listing 4.1 ContractFactory.sol: createContract() Function

```
39    function createContract(uint256 _amount) public payable {
40        require(msg.value == _amount, "Sent value does not match the specified amount.");
41        Contract newContract = new Contract{value: _amount}(msg.sender);
42        userContracts[msg.sender].push(newContract);
43
44        // Increment contract counter and map new contract's address to the counter
45        contractCounter++;
46        contractsByID[contractCounter] = address(newContract);
47
48        // Store the current block's timestamp
49        timestampByID[contractCounter] = block.timestamp;
50
51        // Call registerNewContract with the new contract's address
52        this.registerNewContract(address(newContract));
53
54        emit ContractCreated(msg.sender, newContract, contractCounter);
55    }
```

The helper functions that are contained in the contract factory to help authentication services to better track newly created contracts can be seen in listing 4.2.

Listing 4.2 ContractFactory.sol: Authentication Service Helper Functions

```
57
58      function getContractsByUser(address user) public view returns (Contract[] memory) {
59          return userContracts[user];
60      }
61
62      function getContractByID(uint256 contractID) public view returns (address) {
63          return contractsByID[contractID];
64      }
65
66      // Fetch the timestamp by contract ID
67      function getContractTimestampByID(uint256 contractID) public view returns (uint256) {
68          return timestampByID[contractID];
69      }
```

4.1.2 Ride Contract

Now that we have shown how a ride contract is created through the ride contract factory, it is important to look at the ride contract itself. The Solidity smart contract under consideration allows the ride provider and the customer to interact with each other and tracks these interactions as described in figure 3.1.

The contract starts with the initiation of the variables that are used to track the status of the ride through a constructor, seen in listing 4.3. The constructor also sets the wallet address of the customer who initiated the contract through the contract factory as ”party1”. The address that is mapped to party one represents the customer inside the contract.

Listing 4.3 Contract.sol: Constructor

```
30      constructor(address _party1) payable {
31          party1 = _party1;
32          rideProviderAcceptedStatus = false;
33          rideProviderArrivedAtPickupLocation = false;
34          userReadyToStartRide = false;
35          rideProviderStartedRide = false;
36          rideProviderArrivedAtDropoffLocation = false;
37          userMarkedRideComplete = false;
38          userCanceldRide = false;
39          rideProviderCanceldRide = false;
40
41      }
```

After the contract is created by the customer, the address of the contract is shared via the matching service with the ride provider. The ride provider then uses the `signContract()` function to co-sign the contract as ”party2” and thereby activates the ride contract, seen in listing 4.4. The ride provider also has to deposit 10% of the predefined maximum ride cost into the contract as part of the deposit trust mechanism.

Now that both parties have signed the contract the actual ride flow can start, as decided in figure 3.7 with the ”Ride Provider accepts ride” event. All event functions are structured similarly. Therefore, we will use the `setRideProviderAcceptedStatus()` function, seen in listing 4.5 as an example to showcase how each event is represented by a function inside the smart contract. The function takes a message as an input that will later be written onto the chain permanently. This message is used by the customer and the ride provider to exchange encrypted information on the blockchain as decided via the Diffie-Hellman Key Exchange. The function also

Listing 4.4 Contract.sol: signContract() Function

```
76      function signContract() public payable {
77          require(party2 == address(0), "Party2 has already signed the contract.");
78          require(!isActive, "Contract is already active.");
79          require(!userCanceldRide, "User cannceld ride ");
80          require(msg.sender != party1, "Party2 cannot be identical to Party1.");
81
82          party2 = msg.sender;
83          isActive = true;
84
85          uint256 tenPercent = (address(this).balance * 10) / 100;
86          require(msg.value >= tenPercent, "Party2 must deposit an amount equal to 10% of the contract
    balance.");
87
88          // Refund any excess amount deposited by party2
89          if (msg.value > tenPercent) {
90              payable(msg.sender).transfer(msg.value - tenPercent);
91          }
92      }
```

Listing 4.5 Contract.sol: setRideProviderAcceptedStatus() Function

```
97      function setRideProviderAcceptedStatus(string memory _message) public {
98          require(isActive, "Contract is not active.");
99          require(msg.sender == party2, "Only Party2 can set the ride provider accepted status.");
100         require(!rideProviderAcceptedStatus, "Ride Provider Accepted Status can only be set once.");
101
102         require(!rideProviderCanceldRide, "Ride Provider Canceld Ride Status can only be set once.");
103         require(!userCanceldRide, "User Canceld Ride Status can only be set once.");
104
105         rideProviderAcceptedStatus = true;
106         emit UpdatePosted(msg.sender, _message, "rideProviderAcceptedStatus");
107     }
```

contains a number of checks utilising the `require()` function to make sure that it is only used as intended. Among others, these checks include the requirement to only be called by the right entity (depending on the event, this can be the customer or the ride provider). Additionally, a check is applied to ensure that the events are called in the ride order as designed in the ride flow. At last, three more checks make sure that the contract is active, not cancelled, and a ride provider has accepted the contract. If all checks are successful, it can be ensured that the function is used as intended and the event is written onto the chain as secure and auditable proof that it accrued. This is done by calling the `UpdatePosted()` function.

The `UpdatePosted()` function can be seen in listing 4.6 It gets called by an event function and takes the wallet of the entity calling the function, the message and the name of the event function and emits it as an event onto the chain. These events are permanent and can not be changed.

Listing 4.6 Contract.sol: updatePosted() Event

```
94  event UpdatePosted(address indexed author, string message, string functionName);
```

Listing 4.7 Contract.sol: setUserCanceldRide() Function

```
176     function setUserCanceldRide(string memory _message) public {
177         require(msg.sender == party1, "Only Party1 can set the user cancelled ride status.");
178 
179         if(!isActive) {
180             uint256 balance = address(this).balance;
181             payable(party1).transfer(balance);
182             return;
183         }
184 
185         require(!rideProviderCanceldRide, "Ride Provider Canceld Ride Status can only be set once.");
186         require(!userCanceldRide, "User Canceld Ride Status can only be set once.");
187 
188         userCanceldRide = true;
189 
190         if(isActive) {
191             uint256 balance = address(this).balance;
192             payable(party2).transfer(balance);
193         }
194 
195         emit UpdatePosted(msg.sender, _message, "userCanceldRide");
196     }
```

Listing 4.8 Contract.sol: setRideRating() Function

```
222   function setRideRating(uint _rating) public {
223         require(msg.sender == party1, "Only Party1 can set the ride rating.");
224         require(!isRideRatingSet, "Ride rating can only be set once.");
225         require(_rating >= 0 && _rating <= 5, "Rating must be between 0 and 5.");
226         require(isActive, "Contract is not active.");
227         rideRating = _rating;
228         isRideRatingSet = true;
229     }
```

Now that the functions that enable the general ride flow are shown, it is important to take a look at edge cases that are handled on-chain. As described in figure 3.1, both the customer and ride provider have the ability to cancel the ride and if one of the parties cancels the ride the other party gets automatically deposited the money managed by the ride contract. The ride contract provides these features through a `setUserCanceldRide()` and as `setRideProviderCanceldRide()` function. In the flowing, we will describe their functionality using the `setUserCanceldRide()` function as an example, seen in figure 4.7. If the customer calls the function to cancel their ride, two if clauses check if the contract is active and, therefore a ride provider has already signed the contract. If this is the case, all money inside the contract gets deposited to the ride provider. If for some reason, no ride provider is signing the contract, the customer gets their money back. This ensures that the customer does not get their deposit stuck inside a ride contract if a ride provider decides not to sign a contract. Additionally, the customer has the possibility to emit a message onto the chain that contains information on why the ride was cancelled. Lastly, the contract gets marked with the status `userCanceldRide = true`.

As described in subsection 3.2.2, the rating is also managed via the smart contract. Both customers and ride provider can post their ratings through similar functions. The customer can use the `setRideRating()` function for this process which takes the selected rating as input, as seen in figure 4.8. The function ensures, besides other checks, that the rating is in the predefined range of allowed ratings.

Listing 4.9 Contract.sol: addPassenger() Function

```
53     function addPassenger(string memory _passengerID, uint _seatingPosition, string memory _startTime)
   public {
54         require(isActive, "Contract is not active.");
55         require(msg.sender == party2, "Only Party2 can add passengers.");
56 
57         Passenger memory newPassenger = Passenger({
58             passengerID: _passengerID,
59             seatingPosition: _seatingPosition,
60             startTime: _startTime,
61             rating: 5
62         });
63 
64         passengers.push(newPassenger);
65     }
```

It is also possible for the customer to rate passengers. Therefore the `addPassenger()` function is used by the ride providers to add passengers to the ride contract, as seen in 4.9. This will allow customers to rate their passengers. For privacy reasons, the customer will only see the pseudonyms known to the ride provider for the passengers. The user can then rate the passengers based on their seating position inside the vehicle and the starting time of their ride. Through this system, it is possible for customers sharing a vehicle to rate each other without the need to share personal information like a name or a profile picture.

The `claimETH()` function is used by the ride provider to get the applicable amount of money to cover the ride cost from the smart contract, as seen in figure 4.10. The function can only be triggered once the customer has marked the ride as successfully completed. After all, checks are successfully completed, the contact sends 10% of the total amount of money on hold inside the contract to a predefined address managed by the GETACAR foundation. Afterwards, the remaining money gets deducted from the amount of money that is claimed by the ride provider to cover the ride cost. This amount is then transferred to the wallet of the ride provider. The remaining deposit left inside the smart contract then gets sent back to the wallet of the customer.

4.1.3 Matching Contract

The Matching Contract provides the fully on-chain rating and load-balancing for the off-chain matching services as described in 3.1.6. At its core, the function provided by the Matching contract is simple: A customer can provide an array of matching services and a minimum rating, and the smart contract returns the matching service out of this array that complies with the rating requirements and has the lowest number of handled matches so far. This allows for a load balancing between the matching services so that no service can collect too much data.

All matching services that are available to the GETACAR platform are registered inside the Matching contract with a struct that contains the name of the service, the number of matches that were handled by the matching service (that resulted in completed rides) and the number of requests that where managed by the service, as seen in figure 4.11. To ensure that the request counter is accurate, the contract counts a request every time a specific matching service gets suggested by the Matching Contract, as the design assumes that the customer will follow the suggestion and post their ride request to the suggested matching service. For example, a customer provides an array containing the names of matching services A, B and C without defining a minimum rating. The matching contract determines that matching service A has had the lowest amount of requests out of the three so far and, therefore, suggests the use of matching service A to the customer. This suggestion is counted as a request, and the request counter of matching service A is

Listing 4.10 Contract.sol: claimETH() Function

```
232     function claimETH(uint256 amount) public {
233         require(isActive, "Contract is not active.");
234         require(msg.sender == party2, "Only Party2 can claim the deposited ETH.");
235         require(userMarkedRideComplete, "User must mark the ride complete before claiming the deposited
    ETH.");
236         require(amount <= address(this).balance, "Requested amount exceeds the contract balance.");
237 
238         address payable hardcodedAddress = payable(0xE39a3085CB78341547F30a1C6bD12977d51aa967);  //
    Address of the GETACAR Foundation
239 
240         uint256 balance = address(this).balance;
241         uint256 tenPercent = balance / 10;
242         uint256 remainder = balance - tenPercent;
243 
244         hardcodedAddress.transfer(tenPercent);
245 
246         uint256 payback = remainder - amount;
247         remainder -= payback;
248 
249         payable(party1).transfer(payback);
250         payable(party2).transfer(remainder);
251     }
```

Listing 4.11 Matching.sol: MatchingServiceObject Struct

```
6     struct MatchingServiceObject {
7         string name;
8         uint256 matches;
9         uint256 requests;
10    }
```

increased by one. In the prototype implementation, the load-balancing factor is the number of successful matches managed by the matching service, as this value is hard to manipulate. As an alternative, the number of requests can also be used as the load-balancing factor. Further research is needed to determine which factor is best suited for balancing requests between the matching services.

The actual recommendation function can be seen in listing 4.12. For the prototype, the flow does not support automatic filtering by rating.

After explaining the request counter, it is important to take a look at how the match counter works, as it is equally necessary for calculating the rating of the matching service. What makes the calculation of this value more complicated is the fact that the contract needs to verify that only rides are counted that were officially handled by the GETACAR platform. If this is not the case, it would open up doors for rating manipulations through external parties. To prevent that, the `addMatch()` function is not called by a customer who has successfully completed a ride but can only be called by the ride contracts themselves. It can be seen in listing 4.13 The function utilises a for loop that checks a list that contains all contracts that are created by the official contract factory. If the contract that calls the function is on the list, the value of the utilised matching service is increased by one.

Listing 4.12 Matching.sol: getMatchingService Function

```
59    function getMatchingService(string[] memory names) public {
60        uint256 lowestMatches = type(uint256).max;
61
62        string memory lowestMatchServiceName = "";
63        uint256 lowestMatchServiceRating;
64
65        for (uint i = 0; i < names.length; i++) {
66            for (uint j = 0; j < services.length; j++) {
67                if (keccak256(bytes(services[j].name)) == keccak256(bytes(names[i]))) {
68                    if (services[j].matches < lowestMatches) {
69                        lowestMatches = services[j].matches;
70                        lowestMatchServiceName = services[j].name;
71                        lowestMatchServiceRating = (services[j].matches * 100) / services[j].requests; //
Multiply by 100 for two decimal places
72                        services[j].requests += 1;
73                    }
74                }
75            }
76        }
77        // Emit the event with the result
78        emit LowestMatchService(lowestMatchServiceName, lowestMatchServiceRating);
79    }
```

Listing 4.13 Matching.sol: addMatch() Function

```
81    function addMatch(string memory serviceName) external onlyRegisteredContracts {
82        for (uint i = 0; i < services.length; i++) {
83            if (keccak256(bytes(services[i].name)) == keccak256(bytes(serviceName))) {
84                services[i].matches += 1;
85            }
86        }
87    }
```

To enforce that only the contract factory can add ride contract addresses to the list of verified addresses, an `onlyFactory()` modifier is implemented, as seen in listing 4.14. Similarly, a `onlyRegisteredContracts()` modifier ensures that only contracts from the list of verified contracts can add successful matches to the rating services.

Listing 4.14 Matching.sol: onlyFactory() and onlyRegisteredContracts() Modifier

```
24    modifier onlyFactory() {
25        require(msg.sender == FACTORY_ADDRESS, "Only the factory can call this");
26        _;
27    }
28
29    modifier onlyRegisteredContracts() {
30        require(registeredContracts[msg.sender], "Only registered contracts can call this");
31        _;
32    }
```

4.2 Customer Frontend and Virtual Vehicle

While the smart contracts represent the backbone of GETACAR, it is the frontend for customers and ride providers that enables the users to interact with the platform. The developed prototype provides a fully designed frontend for customers. The ride provider frontend is built as a service daemon called Virtual Vehicle that could be utilised by autonomous vehicles to interact with the platform and atomically interacts with the platform by posting bids on new rides and interacting with the ride contracts.

4.2.1 Customer Frontend Flow

The customer frontend prototype is built using Angular [6] and the web3 package [7] to interact with the Ethereum blockchain and crypto wallets. While the market-ready implementation of a decentralised ride-pooling platform would probably utilise native smartphone apps, the Angular web implementation allows for a fast, platform-independent development to showcase the most important features of the customer ride-pooling frontend.

The customer frontend follows the three-view design approach described in subsection 3.1.3, offering a booking view, an on-ride view and a settings view.

The process of ordering a ride follows the customer flow seen in figure 3.1 by opening the app and entering the destination and pickup location. The customer frontend provides a map that can preview the trip. Additionally, the frontend provides a compass needle button as part of the pickup location input field. Pressing this button allows the customer to select their current location as their pickup location, shown in figure 4.3.

[6] https://angular.io
[7] https://www.npmjs.com/package/web3

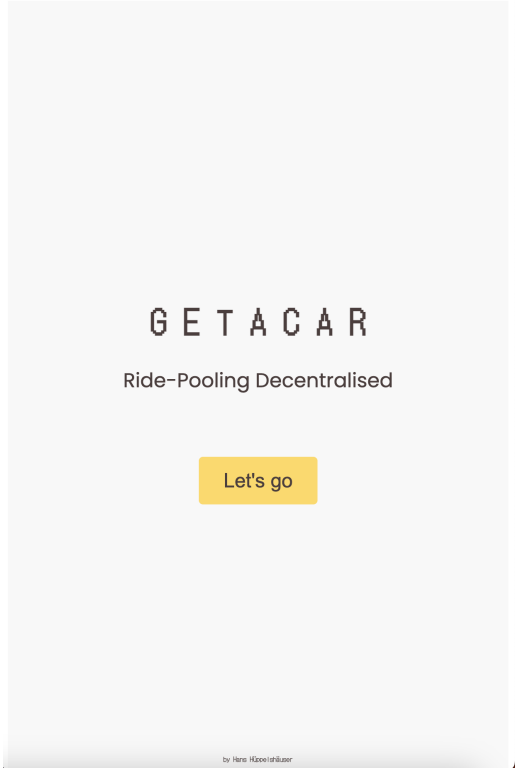

Figure 4.2: Frontend: Welcome Screen

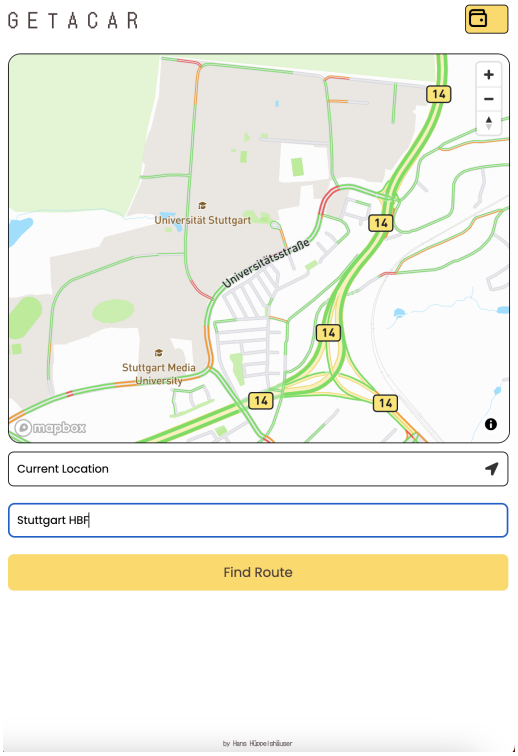

Figure 4.3: Frontend: Map Screen

After the customer presses the ”Find Route” button, they will be presented with a preview of their trip, including some additional information like the expected duration and distance of the travel. The customer now has the possibility to post a ride request by pressing the ”Request Ride” button, as shown in figure 4.4. This will send a ride request to the matching service (that was suggested through the matching smart contract). As long as the auction is running on the matching service, the customer frontend will display the loading screen seen in figure 4.5.

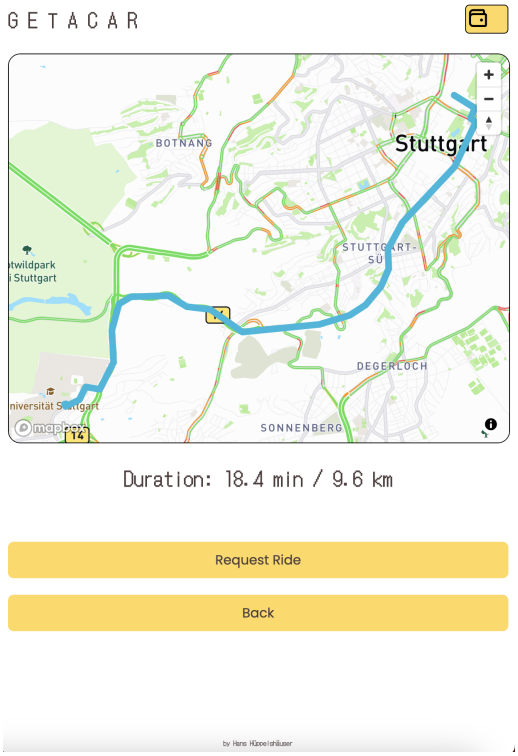

Figure 4.4: Frontend: Map Trip Screen

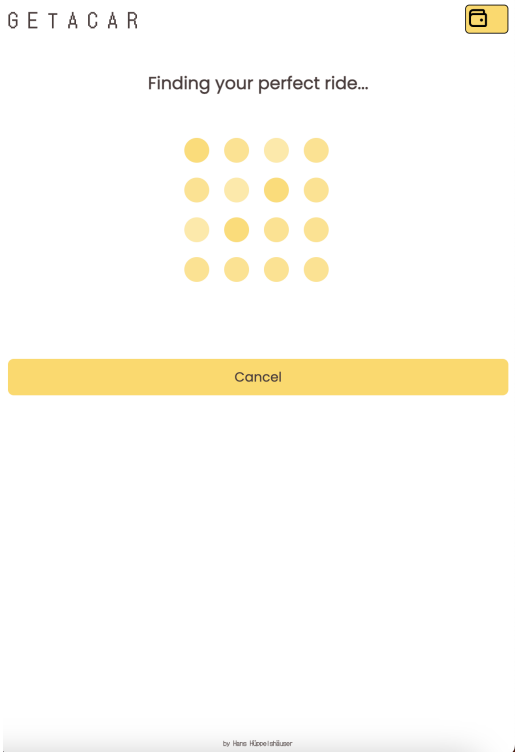

Figure 4.5: Frontend: Search Ride Screen

Once the auction is closed, the customer is presented the winning bid with all the important information about the ride, as shown in figure 4.6. In the top right corner, a timer counts down the time that is left before the offer expires. As long as the offer has not expired, the customer is able to press the ”Book Ride” button to confirm the ride and thereby create a ride contract on the blockchain. The frontend also has dedicated screens that inform the customer if the matching service could not find a ride or if the offer expired because the customer did not press the ”Book Ride” button.

After a ride is found and the customer confirms the ride, the frontend changes into the on-ride view that provides status updates about all activities happening on the ride and tracks them via the ride contract. Figure 4.7 shows the ”waiting Confirmation Screen”. At this stage, the frontend monitors the blockchain and waits for the event from the ride contract that signals that the ride provider has co-signed the ride contract.

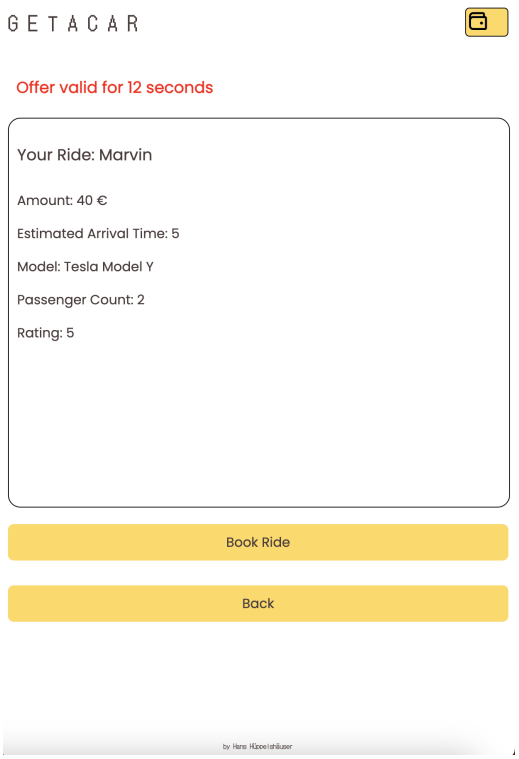

Figure 4.6: Frontend: Ride Overview Screen

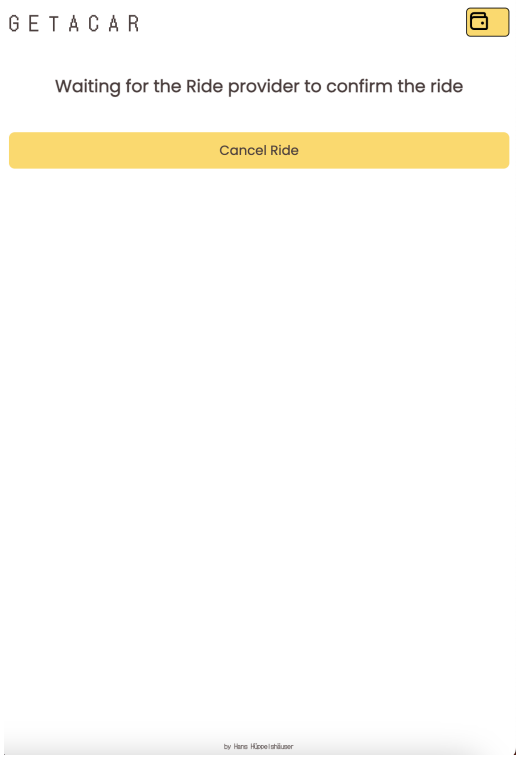

Figure 4.7: Frontend: Awaiting Confirmation Screen

When the ride provider has signed the contract and confirmed that they started driving to the agreed pickup location, the on-ride screen changes to display the new status of the ride, as seen in figure 4.8. A market-ready version of the platform would also show the live position of the ride provider to the customer that the ride provider has shared as an encrypted message through the ride contract. However, this feature is not part of the prototype implementation as the prototype only utilises virtual vehicles as ride providers that are not able to provide real-time GPS position updates.

Once the ride provider has arrived at the pickup location and posted this update on the ride contract, the customer's frontend changes again to display the update, shown in figure 4.9. The customer now enters the vehicle, and once they are ready to start their ride, they confirm this through the "Start Driving" button.

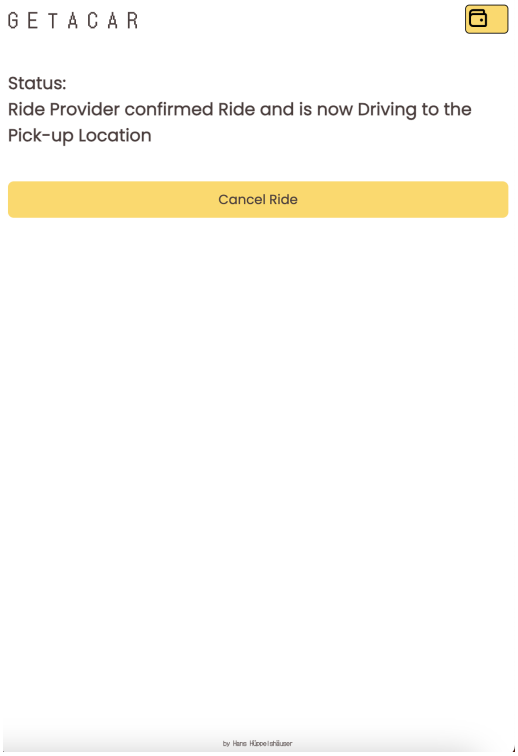

Figure 4.8: Frontend: Pickup Location Drive Screen

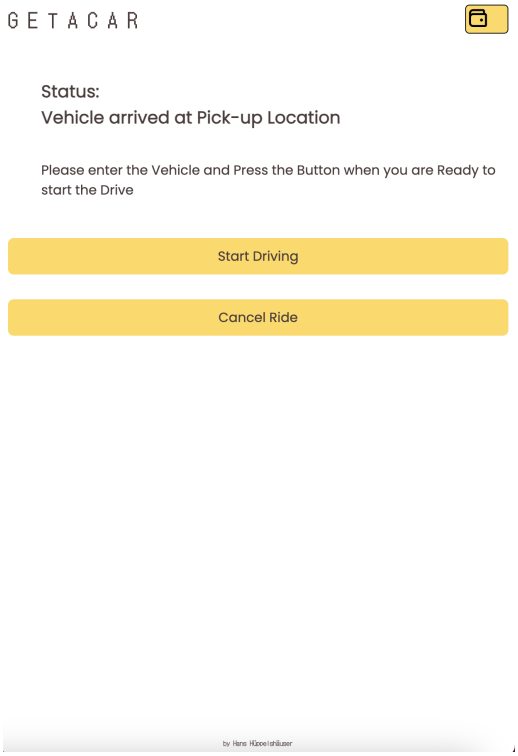

Figure 4.9: Frontend: Vehicle Arrived Screen

After the ride provider has received the update from the customer that they are ready, the ride provider starts the ride to the destination. This is also represented via a status screen in the customer frontend, as shown in figure 4.10. When arriving at the dropoff location, the ride provider posts this information onto the ride contract. The customer frontend notices this new event on the blockchain and updates the status of the frontend, as seen in figure 4.11. On this screen, the customer has the possibility to confirm the ride completion and thereby end the ride process.

It is important to note that each status screen also provides a ”Cancel Ride” button at all times that allows the customer to exit the ride process.

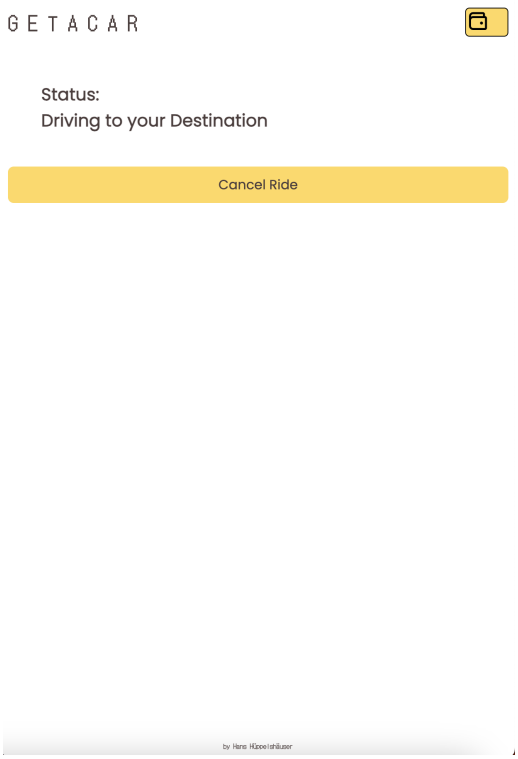

Figure 4.10: Frontend: Driving Screen

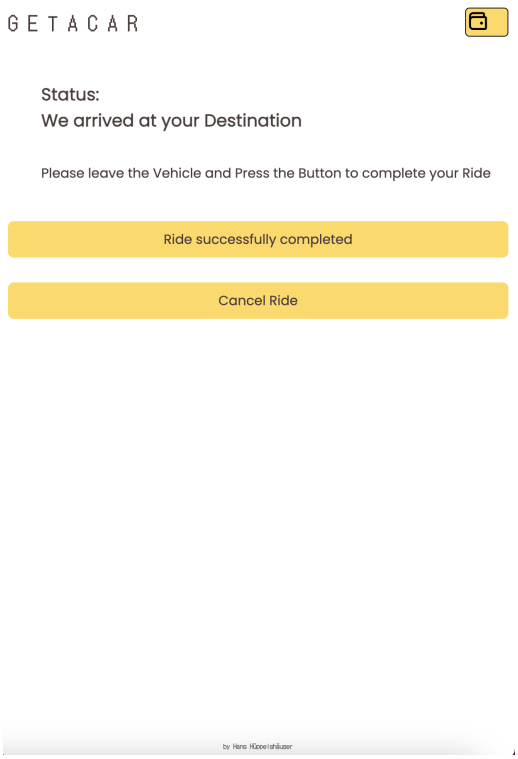

Figure 4.11: Frontend: Destination Screen

Once the ride has ended, the customer is provided with a ”Ride Completed” screen that allows the customer to rate the ride provider itself and their passengers, seen in figure 4.12. As described in subsection 3.2.2, the problem with rating passengers is that no personal information should be exchanged through the platform about the passengers. This makes it complicated to ensure that the customer knows who is who when rating multiple passengers. To solve this problem, the ride provider shares the seating position and the start time of each passenger with the customer. The seating position is then visualised by the frontend, as seen in figure 4.13. In this example, the passenger that is getting rated was sitting on seat number one, which is the seat on the front row on the left.

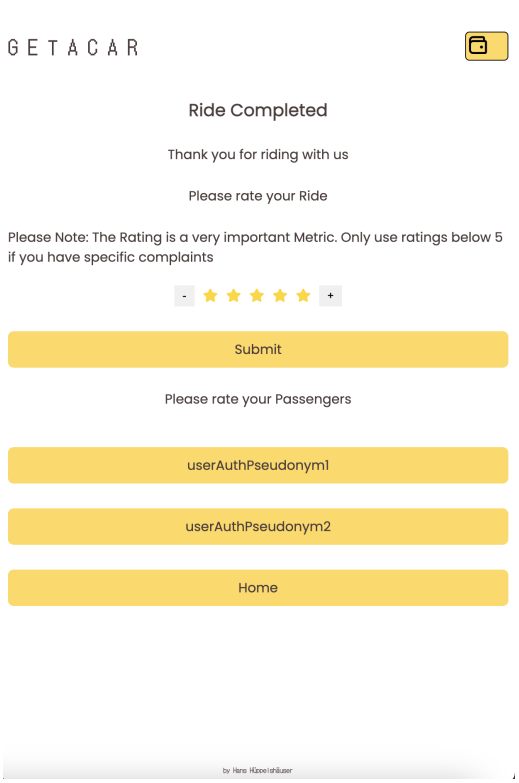

Figure 4.12: Frontend: Rate Ride Provider Screen

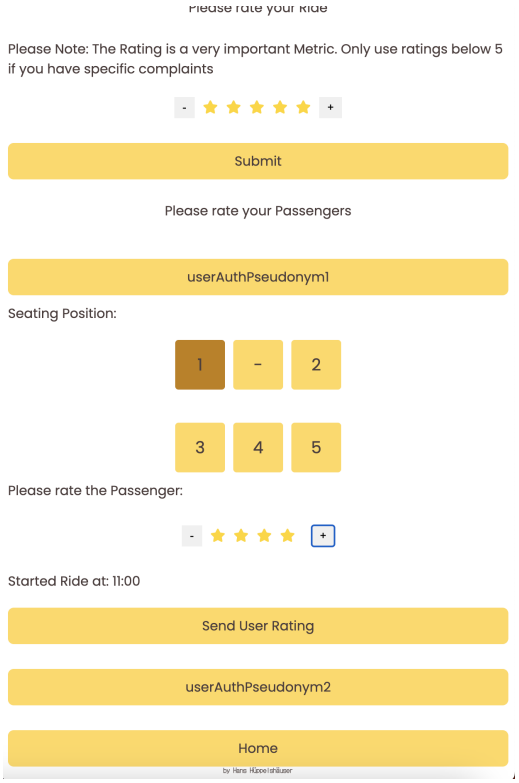

Figure 4.13: Frontend: Rate Passenger Screen

Last but not least, the prototype frontend also contains the settings view as seen in figure 4.14, which can be opened by pressing the wallet icon in the top right corner of the UI. Here, the customer has a number of sliders available to adjust ride booking preferences, like the minimum ride provider rating and the maximum amount of passengers that they want to share the ride with at once. This view also shows the rating of the customer themself as well as some additional information like the wallet id that is used to connect to the platform and the account balance of the wallet. This information is only implemented for the prototype as the market-ready implementation of the frontend would manage multiple wallets at once. One for each ride. The wallets would be managed by the frontend in the background as the customer does not have to manage all the wallets by themselves self and each wallet will only be charged with the amount of money needed for one ride through a crypto exchange.

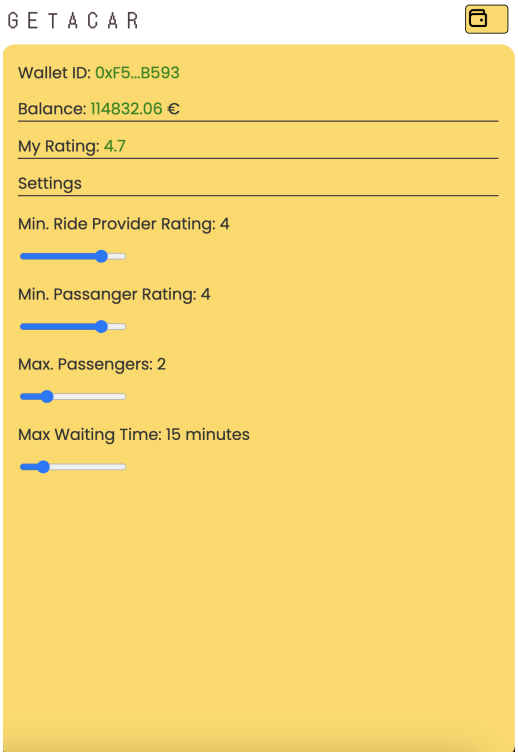

Figure 4.14: Frontend: Settings Screen

4.2.2 Customer Frontend Logic

Now that the user flow of the customer frontend is shown, it is important to take a short look at the code logic behind the UI. While most of the logic behind the frontend is standard angular code, the utilisation of the web3[8] package is notable, as it allows the frontend to interact with smart contracts directly. To demonstrate this interaction, the `setUserReadyToStartRide()` function is chosen as an example. This function gets triggered when a customer presses the "Start Driving" button shown in figure 4.9. First, the function checks if a wallet is connected with the frontend. The prototype utilises the external wallet provider MetaMask[9] to manage multiple wallets. After the check is successful, the function initiates a new, local instance of the contract. Once this is done successfully, the method calls the `setUserReadyToStartRide()` function inside the ride contract with an encrypted `userReadyToStartRideMessage` as input, first to calculate the excepted Gas fee and after the transaction is confirmed through Meta Mask by the user, a second time to actually trigger the `setUserReadyToStartRide()` function inside the ride contract, as seen in listing 4.15.

All functions that are contained within the customer frontend follow this basic flow to interact with the ride contract on the blockchain.

[8] https://www.npmjs.com/package/web3
[9] https://metamask.io

Listing 4.15 booking.component.ts: async setUserReadyToStartRide() Function

```
24  async setUserReadyToStartRide(){
25
26    if (!this.web3) {
27      console.error('Wallet not connected');
28      return;
29    }
30
31    const accounts = await this.web3.eth.getAccounts();
32    const selectedAddress = accounts[0];
33
34    // Initialize the contract instance
35    this.contract = new this.web3.eth.Contract(
36      contractAbi as AbiItem[],
37      this.rideContractAddress,
38    );
39
40    // Call the createContract function
41    const gasEstimate = await this.contract.methods
42      .setUserReadyToStartRide(userReadyToStartRideMessage)
43      .estimateGas({ from: selectedAddress });
44
45    this.contract.methods
46      .setUserReadyToStartRide(userReadyToStartRideMessage)
47      .send({ from: selectedAddress, gas: gasEstimate })
48      .on('transactionHash', (hash: string) => {
49        console.log('Transaction hash:', hash);
50      })
51      .on('receipt', (receipt: any) => {
52        console.log('Transaction receipt events:', receipt);
53      })
54
55      .on('error', (error: Error) => {
56        console.error('Error:', error);
57      });
58  }
```

4.2.3 Virtual Vehicle

The Virtual Vehicle represents a new component introduced for the prototype implementation of the GETACAR platform. The Virtual Vehicle is a software daemon that automatically interacts with the platform as a ride provider. The Virtual Vehicle is written in Node.js and utilises the same web3 package implementation to interact with the blockchain as the customer frontend. It differentiates itself from the customer frontend by not providing a user frontend but by interacting with the platform in an automated way. When starting the Virtual Vehicle, it automatically connects to a predefined matching service and begins to scan for open ride requests. Once it has found a ride request it automatically bids a random amount of money to handle the right. If the Virtual Vehicle wins the auction, it will sign the ride contract and start to follow the predefined ride flow until the right is completed. Once the ride is completed or in case the Virtual Vehicle does not win the auction, it starts to bid on new ride requests. To test the platform and the matching service, it is possible to spin up multiple Virtual Vehicles that all simultaneously interact with the platform. In conclusion, the Virtual Vehicles help with simulating platform traffic and testing the prototypical implementation of the GETACAR platform by acting as virtual ride providers.

4.3 Matching Service

After showcasing the implementation of the smart contracts and the customer and ride provider frontend, the last component that is part of the prototype implementation of the GETACAR platform is the Matching Service. The matching service is written in Node.js[10] and the bids are stored inside MongoDB[11] as a NoSQL database. The design decision of utilising a NoSQL database in connection with the matching service was made for several reasons. NoSQL databases provide high performance and simplify the storage of the ride requests and ride bids that are stored as simple JSON objects. Lastly, there is also no need for complex SQL functions inside the matching service that would promote the usage of an SQL database.

4.3.1 Endpoints and Data

The Node.js application provides four core endpoints for customers and ride providers to handle all interactions with the matching service. Following ride flow, the first endpoint that is typically utilised is the POST `/requestRide` endpoint. The customer frontend utilises this endpoint to post ride requests to the matching service. The request contains the following data points:

userId: The customer pseudonym provided by an authentication service

pickupLocation: The cloaked pickup location

dropoffLocation: A cloaked dropoff location

userRating: The rating of the customer

rating: The rating of the customer requesting the ride

userPublicKey: A newly generated public key from the customer used for the Diffie-Hellman Key Exchange

maxWaitingTime: The maximum time the customer is willing to wait for the arrival of the ride provider

minRating: The minimum rating necessary for a ride provider to have to be allowed to manage the ride

minPassengerRating: The minimum rating for passengers to have to be allowed to share the ride with the customer

maxPassengers: The maximum amount of passengers the customer is willing to have at once

The Matching Service adds the following data points to the ride request and writes them onto the database:

rideRequestId: A unique id to identify the ride request

gridLocation: The grid square from where the ride request came from. This makes it easier for ride provider to find fitting ride requests if a matching service is deployed for a number of different grid squares.

auctionStartedTimestamp: The timestamp represents the moment the data was posted onto the matching service visible for ride providers to bid on the ride request.

auctionStatus: The status of the auction. The auction can have one of four different statuses: 'open', 'determining-winner', 'waiting-for-signature' or 'closed'.

auctionWinner: The winner of the auction. If the winner was not determined yet, this field is empty.

[10] https://nodejs.org/en
[11] https://www.mongodb.com

winningBid: The unique identifier of the winning bid. If the winner was not determined yet, this field is empty.

p: The prime number used for Diffie-Hellman key exchange.

g: The base used for Diffie-Hellman key exchange.

Once the complete ride request is available in the database, it can be read by ride providers. Ride providers can use the GET `/rideRequests` endpoint to receive a JSON object containing all ride requests with open auctions. The ride provider can search this dataset and, once they find a fitting ride request, bid on it. To bid on a ride request, the ride provider can utilise the POST `/bid` endpoint. A bid request contains the following data points:

rideRequestId: The id of the ride request that this bid is for

rideProviderId: The ride provider pseudonym provided by an authentication service

amount: The maximum ride cost that the ride provider is willing to offer the ride for

rating: The rating of the ride provider

model: The model of the vehicle

estimatedArrivalTime: The time to get to the customer pickup location

passengerCount: The number of passengers inside the vehicle when arriving at the pickup location

vehiclePublicKey: A newly generated public key from the ride provider used for the Diffie-Hellman Key Exchange

Before a bid can be posted to the database, the Matching Service compares the timestamp of the bid with the timestamp of the ride request the bid is associated with to ensure that the auction is truly open. For this prototype, the time frame for this is set to 30 seconds. The bid also gets extended with additional data points by the matching service itself before it gets written onto the database. These are the additional data points:

bidId: A unique id to identify the bid

bidPlacedTimestamp: Timestamp representing the moment the bid is posted

With ride requests and associated bids being written to the database, the next step for the matching service is handling the running auction. Each auction is posted with the status ”open”. A function managed by the Matching Service continuously crawls the database for ride requests where the auction is older than 30 seconds. If such an auction is found, the auction status changes from ”open” to ”determining-winner”. A second function then takes the ride request and analyses all bids connected with the request to determine the winning bid based on the principles of the second price auction. The winning bid is then written into the ride request itself, changing the auction status to ”waiting-for-signature”.

The customer is able to check the status of their auction through the GET `/rideRequest/:rideRequestId` endpoint by providing the identifier of their ride request inside the URL. This endpoint returns the status of the auction, and in case a winning bid is found, it additionally returns the winning bid itself. Based on the winning bid, the customer can then decide if they want to take the ride or not. If the customer decides to take the ride, they follow the ride flow and create a ride contract through the contract factory that contains the maximum ride cost as a deposit. They then need to use the GET `setContractAddress/:rideRequestId/:contractAddress` endpoint to update their ride request with the contact’s address on the blockchain. As the creation of the contract is understood as the initial signing of the ride contract, the status of the auction changes from ’waiting-for-signature’ to ’closed’.

The GET `/rideRequest/:rideRequestId` endpoint also enables the ride provider to track the status of the auction. Through the endpoint, the auction winner is able to receive the address of the ride contract and is, therefore, able to co-sign the contract as a ride provider.

4.3.2 Grid System

As described in 3.1.6, the design of the Matching Service includes a map grid that is utilised to assign matching services to specific jurisdiction zones and to cloak the exact pickup and dropoff locations of customers. There are many possible ways to create a map grid that would allow for this use case. The GETACAR prototype implementation uses H3[12], a hexagonal hierarchical geospatial indexing system that provides a predefined grid for the platform to use. The advantage of H3 is that it provides a number of grid resolutions, as each grid is made up of hexagons and pentagons that themself are made up of smaller hexagons and pentagons, as seen in figure 4.15. An algorithm allows one to easily check if a hexagon/ pentagon of a smaller resolution is contained within a hexagon/ pentagon of a higher resolution. This approach would allow GETACAR to utilise dynamic resolutions for the jurisdiction zones and the location cloaking with higher resolutions used for crowded areas with high traffic, like cities and lower resolutions that cover larger, less crowded areas with less traffic. [H3G23] The prototype uses fixed resolutions with Res 9 for the location cloaking and Res 6 for the jurisdiction zones of the matching services. The available resolutions are displayed in table 4.1.

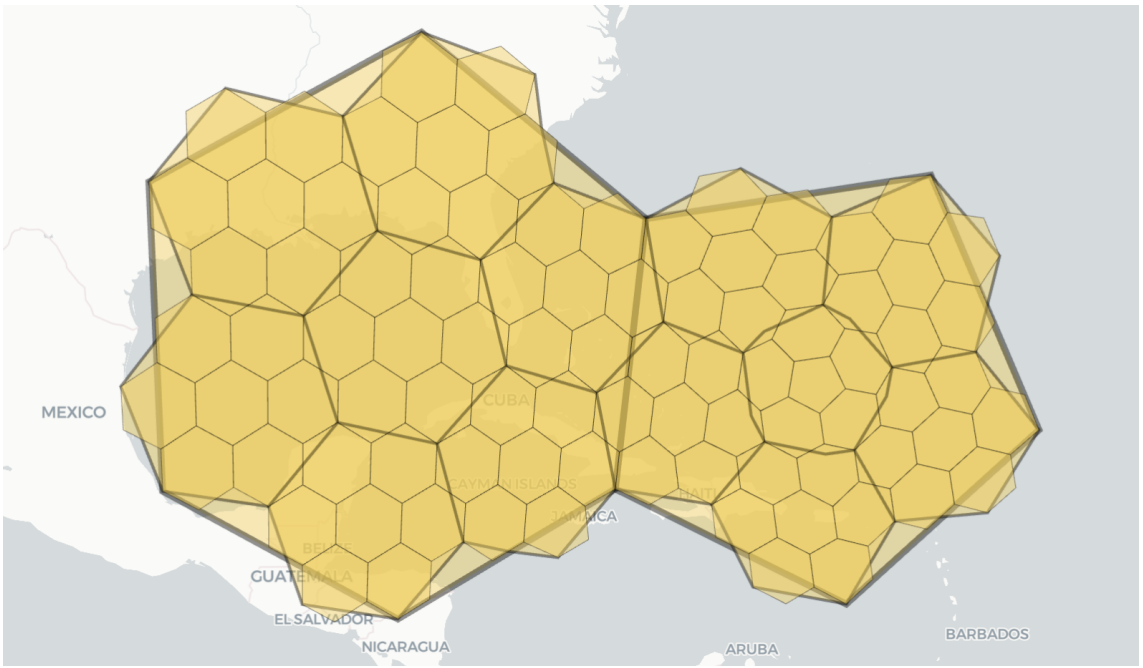

Figure 4.15: H3 Grid Visualisation [H3G23]

[12]https://h3geo.org

Res	Average Hexagon Area (km^2)	Pentagon Area (km^2)	Ratio (P/H)
0	4,357,449.416078381	2,562,182.162955496	0.5880
1	609,788.441794133	328,434.586246469	0.5386
2	86,801.780398997	44,930.898497879	0.5176
3	12,393.434655088	6,315.472267516	0.5096
4	1,770.347654491	896.582383141	0.5064
5	252.903858182	127.785583023	0.5053
6	36.129062164	18.238749548	0.5048
7	5.161293360	2.604669397	0.5047
8	0.737327598	0.372048038	0.5046
9	0.105332513	0.053147195	0.5046
10	0.015047502	0.007592318	0.5046
11	0.002149643	0.001084609	0.5046
12	0.000307092	0.000154944	0.5046
13	0.000043870	0.000022135	0.5046
14	0.000006267	0.000003162	0.5046
15	0.000000895	0.000000452	0.5046

Table 4.1: H3 Grid Resolution [H3G23]

5 Evaluation

After designing the GETACAR platform and developing a prototype based on this design it is important to evaluate the platform to prove its viability. To do so we will validate the GETACAR platform against the objects defined at the beginning of this research in section 1.2. Afterwards, we will conduct a final privacy assessment of the platform and the prototype to ensure that we meet all privacy goals. At last, we will showcase the results from testing the completed GETACAR prototype.

5.1 Validation against Research Objectives

It is important to validate the results of this research against our initial objectives to determine if the research is a success. The following table 5.1 lists all five research objectives that are defined in section 1.2 with the corresponding evaluation.

Table 5.1: Research Objective Assessment

Research Objectives	Description	Evaluation	Fulfilled
Design of the Components and Interaction Flow between the Platform, Customer, and Ride Provider	The research needs to provide a design blueprint for the decentralised ride-pooling platform. This design should communicate the general vision of the platform and explain the key concepts. At its core, the platform is an ecosystem of components interacting with each other. Therefore it is also necessary to design a streamlined, secure, and efficient flow for these interactions. The objective here is to develop an interaction flow that: Ensures Seamless ride booking, facilitates trustworthy transactions and preserves privacy.	Section 3.1 providers a detailed overview of the platform, including a component overview and detailed descriptions of the design of the customer frontend, ride provider frontend, authentication service, matching service, the ride contracts and the integration of crypto exchanges. The interaction flow for each component is described, and additional diagrams are provided. By basing the overall customer and ride provider flow on already established and tested flows while also utilising the advantages of blockchain technology, GETACAR provides seamless ride booking, while also facilitating trustworthy transactions and preserving privacy throughout the platform.	Yes

Continued on next page

Table 5.1 – continued from previous page

Research Objectives	Description	Evaluation	Fulfilled
Design of a Trust Mechanism for Customers and Ride-Providers	Trust is a necessity for every platform but is especially relevant for decentralised platforms as these platforms are not managed by a single owner that can single-handedly settle disputes or resolve unexpected edge cases. Therefore, it is necessary for the platform to have a robust trust system that sanctions malicious behaviour and promotes good behaviour.	The GETACAR platform design includes a detailed design of the trust mechanism, as described in section 3.2. This includes the rating trust mechanisms that allow customers to rate their ride providers and passengers as well as allowing ride providers to rate their customers. All ratings are posted anonymously, and every user of the platform can request ratings from other users based on their pseudonyms. This method allows transparent ratings throughout the platform without exposing the identities of the users. Additionally, the platform utilises deposit trust mechanisms that ensure that monetary incentives are in place for all parties to act according to the predefined ride flow.	Yes
Evaluation of Customer and Ride-Provider Anonymity and Privacy throughout the Platform	One of the disadvantages of blockchain-based platforms is that the high level of transparency can result in a neglect of customer and ride-provider anonymity and privacy. This counts especially for ride pooling platforms where large amounts of personal data like location and transaction data get exchanged. That is why it is important to assess the platform design regarding privacy and anonymity to show that no entity can collect critical amounts of data from the platform.	Ensuring ride-provider anonymity and privacy is especially important on a blockchain-based platform as all transactions are publicly visible. Therefore GETACAR implements robust privacy mechanisms, as described in section 3.2. GETACAR utilises authentication services that provide pseudonyms for users and verifies newly generated wallet addresses. This allows ride providers and customers to take on new identities any time they interact with the platform, be it on-chain or off-chain. This system provides a solid foundation for ensuring that the identities of users are not exposed when interacting with the platform	Yes

Continued on next page

Table 5.1 – continued from previous page

Research Objectives	Description	Evaluation	Fulfilled
Proposal of Solutions for Physical Issues and Edge Cases	While the general focus of the research lies in creating digital processes that allow handling as much of the user flow through the platform as possible, it is important to also design solutions for potential damage to vehicles by passengers or inappropriate actions by individuals towards other passengers that need to be handled outside the platform. Therefore, the aim is to ensure accountability and conceptualise reporting mechanisms.	Ensuring privacy throughout the platform while also enforcing accountability for all users is a complex endeavour. The GETACAR platform solves this problem by utilising authentication services that are the only entities inside the platform that can match pseudonyms and wallet addresses to their owners as described in subsection 3.1.5. To ensure decentralisation, everyone can get verified by the GETACAR Foundation to host an authentication service instance. Thereby, each authentication service only holds a subset of all users registered on the platform. This system enables flows for solving physical issues on vehicles or other edge cases, like if a ride provider wants to make an insurance claim because a customer has damaged a vehicle, they can do so by providing the pseudonym of the user to the authentication service with a request for revealing the identity to the insurance company.	Yes
Prototypical Realisation of the Decentralised Platform	Based on the theoretical design, a prototype implementation of the platform is constructed. By building the platform, it is possible to simulate real-world scenarios, understand unforeseen challenges, and refine the design in response to them.	The design of the GETACAR platform is verified through a prototype that showcases the core functions of the GETACAR platform and the interaction between the components. The prototypical realisation is showcased in chapter 4.	Yes

5.2 Privacy Considerations

Insuring privacy throughout the platform is one of the most important aspects of GETACAR. While we showed the general spread of user information across the components of the platform, as shown in table3.1, it is important to put the platform through a privacy assessment. This assessment is meant to ensure that the privacy design at its core does not contain any loopholes that could endanger user privacy. OMAR et al. provide such an assessment for an anonymity-oriented privacy-preserving reputation system, as it is implemented into GETACAR [HBB22].

GETACAR fulfils the requirements for this assessment as the true identity of users are hidden on the platform, interactions stay anonymous, users are represented by multiple pseudonyms, and transactions can get carried out anonymously.

The paper itself focuses on the privacy preservation of rating systems, but the assessment itself works on a platform level, as the same systems that preserve privacy for users when they interact with the rating systems are also in place for all other interactions on GETACAR. Table 5.2 shows all twelve points of assessment and how the GETACAR platform is evaluated for each.

Table 5.2: User Anonymity-Oriented Privacy-Preserving Reputation System Properties [HBB22]

Property	Description	Evaluation	Fulfilled
Multiple Pseudonyms	A user can assume multiple pseudonyms, either per context or per transaction.	Every user can take on a new pseudonym for each new transaction. For off-chain interactions, the authentication service provides the pseudonyms directly; for on-chain transactions, the user is able to generate their own new wallet, which then gets registered with the authentication service.	Yes
User-Pseudonym Unlinkability	The true identity of a user is not linkable to any pseudonym they use.	By knowing the identity of a user, it is not possible to identify pseudonyms that belong to the user as there is no information contained in the pseudonym that would allow us to make this connection.	Yes
Pseudonym-Pseudonym Unlinkability	Two different pseudonyms of the same user cannot be linked.	The pseudonyms are not linked directly to each other. Therefore, it is not possible to conclude which pseudonyms belong to the same user.	Yes
Rater Anonymity	A user can rate another user without revealing their true identity.	On GETACAR the rater stays anonymous as they use a newly generated wallet as their pseudonym for the ride flow and to post their rating on the blockchain	Yes
Ratee Anonymity	A user can receive a rating without their real identity being disclosed.	GETACAR also provides Ratee anonymity, as users only rate other users based on their wallet pseudonyms on-chain.	Yes
Inquirer Anonymity	A user can inquire about another user's reputation anonymously.	Every user can request the rating of another user without exposing their identity. To get a rating, it is only necessary to provide the pseudonym of the user to the authentication service. The authentication service can then return the rating of the user connected to the pseudonym.	Yes
Reputation Transfer and Aggregation	A ratee can transfer and aggregate reputation among their pseudonyms.	As each pseudonym is connected to a single user by the authentication service, the rating transfers between all pseudonyms	Yes

Continued on next page

Table 5.2 – continued from previous page

Property	Description	Evaluation	Fulfilled
Reputation Unforgeability	A ratee cannot show reputation higher than their pseudonyms' cumulative reputation.	Reputation forgeability is not possible as a user does not provide their rating themself but through the authentication service that represents a trusted authority	Yes
Distinctness	Reputation of a ratee is an aggregate of votes from distinct raters.	As all ratings are posted to a smart contract on the blockchain that logs the rater and ratee and verifies that both parties are part of the platform, distinctness of ratings is ensured.	Yes
Accountability	Users are accountable for adversarial actions.	The rating systems keep users accountable for their actions and allow for the revelation of the identity of a user in edge cases.	Yes
Authorizability of Ratings	Only users who have had a transaction with the ratee are allowed to rate her.	The smart contracts ensure that all ratings are valid, as customers and ride providers can only rate each other after signing a ride contract on-chain.	Yes
Verifiability by Ratee	A ratee should be able to identify all published feedback linked to their identity and verify their authenticity.	As each user knows all their pseudonyms themselves, they can calculate their rating on their own to verify the calculations of the authentication service.	Yes

As seen in the table 5.2 GETACAR is able to satisfy all properties of the assessment. While the fulfilment of some of these points relies on the implementation of the authentication service, which is not completely realised in this paper, this is still a very notable achievement, as the paper [HBB22] does not identify a single source out of 26 analysed research papers, that fulfils all of the properties of the assessment.

5.3 Testing and Results

After assessing the research objectives and the security measures of the GETACAR platform, it is important to also document the results of prototype implementation.

The prototype shows that the design of the frontends, the matching service, and the smart contracts successfully translates into a working implementation. All components work in tandem, and it is possible to complete the ride-booking flow. The simulation was conducted with one frontend and ten virtual vehicles bidding on the ride request. The matching service successfully determined the winner of the Vickrey auction, and the frontend was able to generate a ride contract from the contract factory based on the winning bid. The virtual vehicle with the winning bid co-signed the contract, and the ride flow itself was tracked through the smart contract, including the deposit payout and the rating posted by the ride provider and customer.

The smart contracts are deployed on the Ethereum node simulator Ganache. The Gas fees, which represent the resulting computation cost of an exemplary ride when interacting with the smart contract, can be seen in table 5.3 and 5.4. It is important to note that the smart contracts are not optimised for reduced Gas cost and that the cost of each transaction may change depending on factors like the length of the encrypted message provided as part of the emitted events on-chain.

Transaction	Gas Cost
Determining best Matching Service for current Hexagon/Pentagon	7341.82
Creation of Ride Contract on-chain from the Contract Factory	479152.27
Posting the ”Start driving” Event	3915.45
Posting the ”Ride completed” Event	6834.56
Rating Ride Provider	8025.45
Rating one Passenger	5778.64
Sum	**511048.19**

Table 5.3: Gas consumption for the Ride Customer transactions

Transaction	Gas Cost
Co-Signing Ride Contract	2458.82
Posting the ”Accepted Ride” Event	1458.86
Adding first passenger	5396.68
Adding second passenger	4619.41
Posting the ”Arrived at pickup location” Event	1482.50
Posting the ”Ride-provider started ride” Event	1470.18
Posting the ”Ride-provider arrived at dropoff location” Event	1483.55
Claiming Deposit	2414.77
Sum	**20784.77**

Table 5.4: Gas consumption for the Ride Provider transactions

To put these Gas prices into context, we calculate the total cost of completing the ride flow for the customer on the Ethereum blockchain based on the current Ethereum prices available at the time of writing. [1]

Given:

- Gas Cost: $G = 511048.19$
- Gas Price: $P = 20$ Gwei (where 1 Gwei = 10^{-9} ETH)
- Price of 1 ETH in USD: $R = \$1649.36$

The total cost in Gwei for executing the smart contract is calculated as:

$$C_{\text{Gwei}} = G \times P$$

$$C_{\text{Gwei}} = 511048.19 \times 20$$

$$C_{\text{Gwei}} = 10220963.8\text{Gwei}$$

To convert the cost from Gwei to ETH:

$$C_{\text{ETH}} = \frac{C_{\text{Gwei}}}{1,000,000,000}$$

[1] https://etherscan.io/gastracker

$$C_{\text{ETH}} = \frac{10220963.8}{1,000,000,000}$$
$$C_{\text{ETH}} = 0.0102209638\text{ETH}$$

Finally, to calculate the cost in USD:

$$C_{\text{USD}} = C_{\text{ETH}} \times R$$
$$C_{\text{USD}} = 0.0102209638 \times 1649.36$$
$$C_{\text{USD}} \approx \$16.86$$

Thus, the total cost of executing the smart contract on the Ethereum blockchain, given the provided parameters, is approximately $16.86 USD. While $16.86 USD transaction costs for the customer would be too expensive to make the platform economically feasible, it is important to point out that the Ethereum price frequently undergoes strong fluctuations and that Ethereum is, in general, a very expensive chain for running Solidity smart contracts. A more fitting blockchain for hosting the GETACAR smart contracts could be Avalanche, which provides high-speed smart contract transactions at a low price. Based on the Avalanche Gas prices available at the time of writing, the calculation looks as follows[2]:

Given:

- Gas Cost, $G = 511048.19$
- Gas Price, $P = 26$ nAVAX
- Price of 1 AVAX in USD, $R = \$9.27$

The total cost in nAVAX for executing the smart contract is calculated as:

$$C_{\text{nAVAX}} = G \times P$$
$$C_{\text{nAVAX}} = 511048.19 \times 26$$
$$C_{\text{nAVAX}} = 13287253.14\text{nAVAX}$$

To convert the cost from nAVAX to AVAX:

$$C_{\text{AVAX}} = \frac{C_{\text{nAVAX}}}{1,000,000,000}$$
$$C_{\text{AVAX}} = \frac{13287253.14}{1,000,000,000}$$
$$C_{\text{AVAX}} = 0.01328725314\text{AVAX}$$

Finally, to calculate the cost in USD:

$$C_{\text{USD}} = C_{\text{AVAX}} \times R$$
$$C_{\text{USD}} = 0.01328725314 \times 9.27$$
$$C_{\text{USD}} \approx \$0.12$$

[2]https://tokentool.bitbond.com/gas-price/avalanche

Therefore, the total cost of executing the smart contract on the Avalanche blockchain, given the provided parameters, is approximately $0.12 USD. Calculating the maximum amount of platform operations costs that would still allow for a feasible business case and determining the best blockchain for running a decentralised ride-pooling platform in production has to be part of future research.

Another point of future research has to prove that widespread utilisation of GETACAR actually results in a reduction in road traffic. A fitting tool for this kind of simulation is Simulation of Urban MObility (SUMO), a traffic simulation software that is able to handle large networks. To assist this future research, we converted the road network of San Francisco into a SUMO map, as seen in 5.2.

Figure 5.1: SUMO Map: San Francisco, California

In addition, we created a simulation for randomised traffic on the map, as seen in 5.1. When connecting the API interface provided by the SUMO simulation with the GETACAR customer frontend and the virtual vehicles, future researchers can simulate the effects of the GETACAR ride-pooling platform on large road networks.

Figure 5.2: SUMO Traffic Simulation

6 Conclusion & Outlook

With an expected increase in overall traffic in the near future caused by the widespread adaptation of autonomous vehicles, ride-pooling solutions are needed to decrease the number of individual trips on the road. As current ride-pooling platforms have shown to be insufficient for tackling this problem because of their centralised nature and how they deal with data privacy, this paper introduces GETACAR, a decentralised, privacy-preserving ride-pooling platform. The design of the GETACAR ride-pooling platform is based on a comprehensive literature analysis that outlines the current state and shortcomings of decentralised ride-pooling platforms in scientific research. The design of the GETACAR platform combines the findings from the research analysis with best practices from the industry and contributes its own design suggestions to the discourse. The design of the GETACAR platform demonstrates how blockchain can be used to track rides, how the interaction flow between customers and ride providers is supposed to look like and how payments and service fees can be managed throughout the platform. Additionally, the design covers concepts that ensure privacy and anonymity for all users as well as decentralised trust mechanisms. GETACAR also describes how security is ensured throughout the platform and how edge cases can be handled off-chain.

The design of GETACAR is verified through the creation of a prototype that showcases all relevant features of the platform and allows a simulation of the complete ride flow between the customer, platform and ride provider. The prototype proves the feasibility of the GETACAR platform design and decentralised, privacy-preserving ride-pooling in general.

Therefore, the findings gained in this work can be used as a basis for future research. The design can be used as a blueprint to create market-ready, privacy-preserving ride-pooling platforms that favour user interests over corporate gains. We hope that GETACAR contributes to the research landscape of decentralised ride-pooling and, therefore, helps to solve the problem of increased traffic caused by autonomous vehicles.

6.1 Limitations

While the GETACAR platform design provides an in-depth overview of all relevant components necessary to create a decentralised privacy-preserving ride-pooling platform, it is essential to note that the creation of such a platform is highly complex, and the area still provides room for improvement. First of all, smart contracts, the platform's backbone, are written highly verbose to better illustrate the key functions of the contracts. Optimising the contracts will result in lower Gas fees, which decrease the operation cost of the platform overall. Secondly, while the platform is built with decentralised authentication services in mind, the prototype does not include the component. There is also no mash of crypto exchanges connected with the prototype, allowing customers and ride providers to use fiat currency on the platform. These factors currently limit the proof of feasibility. While the on-chain interaction flow is worked out in detail, there is no predefined communication protocol for the encrypted interaction between customer and ride provider, enabled through the Diffie-Hellman Key Exchange. The technical implementation for exchanging encrypted messages is fully realised, but no standardised format for these messages has been defined yet. Lastly, it is important to note that the privacy and security aspects are verified on an architecture and general platform design level. Professional penetration testing of the prototype is needed to fully verify these aspects of the platform.

6.2 Outlook

By providing a fully developed platform design and a functional prototype, GETACAR lends itself to future research in the area of decentralised ride-pooling. A point of interest for future research should be the continuation of the quantitative and qualitative validation of the GETACAR platform design. The quantitative evaluation can be done by creating complex simulations on top of the platform through SUMO. The SUMO simulations should be able to mimic customer and ride provider behaviour at a large scale to evaluate if the platform is able to process these vast amounts of data without running into bottlenecks. While the design of the platform, customer and ride provider flow is heavily based on the findings from the scientific literature, it should be further verified through qualitative testing. Therefore, the platform prototype should be given out to potential customers and ride providers for real-world testing and further refinement of the platform.

Further verifying the design of the GETACAR platform is important for the creation of a successful, market-ready decentralised ride-pooling platform, but there are also other research topics related to the creation of such a platform that have not been covered in the paper. This includes a detailed legal analysis of how the GETACAR Foundation should be set up, how the foundation should handle tasks like assigning the matching services to the world grid, and how the foundation's process should look for verifying new authentication services. While we try to ensure basic economic feasibility for the platform, detailed research on this topic is needed to develop a detailed business case for GETACAR.

Bibliography

[ANWK21] A. Alkhalifah, A. Ng, P. A. Watters, A. S. M. Kayes. “A Mechanism to Detect and Prevent Ethereum Blockchain Smart Contract Reentrancy Attacks”. In: *Frontiers in Computer Science* 3 (2021). DOI: 10.3389/fcomp.2021.598780 (cit. on p. 25).

[BBA+21] M. M. Badr, M. Baza, S. Abdelfattah, M. Mahmoud, W. Alasmary. “Blockchain-Based Ride-Sharing System with Accurate Matching and Privacy-Preservation”. In: 2021, pp. 1–8. DOI: 10.1109/ISNCC52172.2021.9615661 (cit. on pp. 17, 33, 36).

[BFG+15] W. Bandara, E. Furtmueller, E. Gorbacheva, S. Miskon, J. Beekhuyzen. “Achieving Rigor in Literature Reviews: Insights from Qualitative Data Analysis and Tool-Support”. In: *Communications of the Association for Information Systems* 37 (2015). DOI: 10.17705/1CAIS.03708 (cit. on p. 30).

[BFJ20] L. A. D. Bathen, G. H. Flores, D. Jadav. “RiderS: Towards a Privacy-Aware Decentralized Self-Driving Ride-Sharing Ecosystem”. In: 2020, pp. 32–41. DOI: 10.1109/DAPPS49028.2020.00004 (cit. on pp. 32, 34).

[BKB21] S. Banupriya, K. Kottursamy, A. K. Bashir. “Privacy-preserving hierarchical deterministic key generation based on a lattice of rings in public blockchain”. In: *Peer-to-Peer Networking and Applications* 14.5 (2021), pp. 2813–2825. ISSN: 1936-6442. DOI: 10.1007/s12083-021-01117-2 (cit. on p. 23).

[BLM+21] M. Baza, N. Lasla, M. M. E. A. Mahmoud, G. Srivastava, M. Abdallah. “B-Ride: Ride Sharing With Privacy-Preservation, Trust and Fair Payment Atop Public Blockchain”. In: *IEEE Transactions on Network Science and Engineering* 8.2 (2021), pp. 1214–1229. DOI: 10.1109/TNSE.2019.2959230 (cit. on pp. 32, 33, 59).

[BMS+20] M. Baza, M. Mahmoud, G. Srivastava, W. Alasmary, M. Younis. “A Light Blockchain-Powered Privacy-Preserving Organization Scheme for Ride Sharing Services”. In: *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*. IEEE, 5/25/2020 - 5/28/2020, pp. 1–6. ISBN: 978-1-7281-5207-3. DOI: 10.1109/VTC2020-Spring48590.2020.9129197 (cit. on pp. 32, 34, 35, 59).

[CC19] S. E. Chang, C.-Y. Chang. “Application of Blockchain Technology to Smart City Service: A Case of Ridesharing”. In: 2019, pp. 664–671. DOI: 10.1109/Cybermatics/2018.2018.00134 (cit. on pp. 32, 34).

[CSFS20] Q. Covert, D. Steinhagen, M. Francis, K. Streff. “Towards a Triad for Data Privacy”. In: *Proceedings of the 53rd Hawaii International Conference on System Sciences*. Ed. by T. Bui. Proceedings of the Annual Hawaii International Conference on System Sciences. Hawaii International Conference on System Sciences, 2020. DOI: 10.24251/HICSS.2020.535 (cit. on p. 27).

[CSJ+17] D. Conte de Leon, A. Q. Stalick, A. A. Jillepalli, M. A. Haney, F. T. Sheldon. "Blockchain: properties and misconceptions". In: *Asia Pacific Journal of Innovation and Entrepreneurship* 11.3 (2017), pp. 286–300. ISSN: 2071-1395. DOI: 10.1108/APJIE-12-2017-034 (cit. on p. 23).

[DH76] W. Diffie, M. Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638 (cit. on p. 28).

[GBAC21] B. C. Ghosh, T. Bhartia, S. K. Addya, S. Chakraborty. "Leveraging Public-Private Blockchain Interoperability for Closed Consortium Interfacing". In: 99 (2021), pp. 1–10. DOI: 10.1109/INFOCOM42981.2021.9488683. URL: http://arxiv.org/pdf/2104.09801v1 (cit. on p. 23).

[GBE+18] A. E. Gencer, S. Basu, I. Eyal, R. van Renesse, E. G. Sirer. "Decentralization in Bitcoin and Ethereum Networks". In: (2018). DOI: 10.48550/ARXIV.1801.03998 (cit. on p. 23).

[GCZ+22] R. Guo, W. Chen, L. Zhang, G. Wang, H. Chen. "Smart Contract Vulnerability Detection Model Based on Siamese Network (SCVSN): A Case Study of Reentrancy Vulnerability". In: *Energies* 15.24 (2022), p. 9642. DOI: 10.3390/en15249642 (cit. on p. 25).

[H3G23] H3Geo. 2023. URL: https://h3geo.org/ (visited on 09/23/2023) (cit. on pp. 79, 80).

[HBB22] O. Hasan, L. Brunie, E. Bertino. "Privacy-Preserving Reputation Systems Based on Blockchain and Other Cryptographic Building Blocks: A Survey". In: *ACM Comput. Surv.* 55.2 (Jan. 2022). ISSN: 0360-0300. DOI: 10.1145/3490236. URL: https://doi.org/10.1145/3490236 (cit. on pp. 83–85).

[HMS22] S. Hacohen, O. Medina, S. Shoval. "Autonomous Driving: A Survey of Technological Gaps Using Google Scholar and Web of Science Trend Analysis". In: *IEEE Transactions on Intelligent Transportation Systems* 23.11 (2022), pp. 21241–21258. ISSN: 1524-9050. DOI: 10.1109/TITS.2022.3172442 (cit. on pp. 21, 22).

[HSY+22] H. R. Hasan, K. Salah, I. Yaqoob, R. Jayaraman, S. Pesic, M. Omar. "Trustworthy IoT Data Streaming Using Blockchain and IPFS". In: *IEEE Access* 10 (2022), pp. 17707–17721. DOI: 10.1109/ACCESS.2022.3149312 (cit. on pp. 26, 27).

[Ini23] O. S. Initiative. 2023. URL: https://opensource.org (visited on 09/23/2023) (cit. on p. 42).

[JSD+21] R. Joseph, R. Sah, A. Date, P. Rane, A. Chugh. "BlockWheels - A Peer to Peer Ridesharing Network". In: 2021, pp. 166–171. DOI: 10.1109/ICICCS51141.2021.9432188 (cit. on pp. 32, 34, 35).

[KL22] M. C. Kus, A. Levi. "Investigation and Application of Differential Privacy in Bitcoin". In: *IEEE Access* 10 (2022), pp. 25534–25554. DOI: 10.1109/ACCESS.2022.3151784 (cit. on p. 26).

[KUC21] V. KUCHKOVSKY. "BLOCKCHAIN SYSTEM CONSENSUS ALGORITHMS". In: *HERALD OF KHMELNYTSKYI NATIONAL UNIVERSITY* 297.3 (2021), pp. 30–33. ISSN: 23075732. DOI: 10.31891/2307-5732-2021-297-3-30-33 (cit. on p. 24).

[LPZ+23] S. Lu, J. Pei, R. Zhao, X. Yu, X. Zhang, J. Li, G. Yang. "CCIO: A Cross-Chain Interoperability Approach for Consortium Blockchains Based on Oracle". In: *Sensors (Basel, Switzerland)* 23.4 (2023). DOI: 10.3390/s23041864 (cit. on p. 23).

[LWW+19] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, B. He. "A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection". In: (2019). DOI: 10.48550/ARXIV.1907.09693 (cit. on p. 27).

[Lyf23] Lyft. 2023. URL: https://www.lyft.com (visited on 09/23/2023) (cit. on p. 39).

[MAA22] N. Mahmoud, A. Aly, H. Abdelkader. "Enhancing Blockchain-based Ride-Sharing Services using IPFS". In: *Intelligent Systems with Applications* 16 (2022), p. 200135. ISSN: 26673053. DOI: 10.1016/j.iswa.2022.200135 (cit. on pp. 18, 32, 34, 35, 59).

[MLC+22] D. Maffiola, S. Longari, M. Carminati, M. Tanelli, S. Zanero. "GOLIATH: A Decentralized Framework for Data Collection in Intelligent Transportation Systems". In: *IEEE Transactions on Intelligent Transportation Systems* 23.8 (2022), pp. 13372–13385. ISSN: 1524-9050. DOI: 10.1109/TITS.2021.3123824 (cit. on p. 26).

[MTD21] D. Mitra, L. Tauz, L. Dolecek. *Overcoming Data Availability Attacks in Blockchain Systems: LDPC Code Design for Coded Merkle Tree*. 2021. DOI: 10.36227/techrxiv.16532853.v1 (cit. on p. 24).

[Nak09] S. Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". In: *Cryptography Mailing list at https://metzdowd.com* (2009) (cit. on p. 23).

[NDT20] T. Ncube, N. Dlodlo, A. Terzoli. "Private Blockchain Networks: A Solution for Data Privacy". In: *2020 2nd International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*. 2020, pp. 1–8. DOI: 10.1109/IMITEC50163.2020.9334132 (cit. on p. 28).

[NIR+23] R. Neiheiser, G. Inácio, L. Rech, C. Montez, M. Matos, L. Rodrigues. "Practical Limitations of Ethereum's Layer-2". In: *IEEE Access* 11 (2023), pp. 8651–8662. DOI: 10.1109/ACCESS.2023.3237897 (cit. on p. 55).

[Nzu19] S. Nzuva. "Smart Contracts Implementation, Applications, Benefits, and Limitations". In: *Journal of Information Engineering and Applications* (Oct. 2019). DOI: 10.7176/JIEA/9-5-07 (cit. on p. 25).

[PDKJ22] G. D. Putra, V. Dedeoglu, S. S. Kanhere, R. Jurdak. "Blockchain for Trust and Reputation Management in Cyber-Physical Systems". In: vol. 194. 2022, pp. 339–362. DOI: 10.1007/978-3-031-07535-310 (cit. on p. 28).

[PHSB21] N. Périvier, C. Hssaine, S. Samaranayake, S. Banerjee. "Real-time Approximate Routing for Smart Transit Systems". In: 2021, pp. 73–74. DOI: 10.1145/3410220.3460096 (cit. on p. 22).

[Pie21] G. A. Pierro. "Smart-Graph: Graphical Representations for Smart Contract on the Ethereum Blockchain". In: 2021, pp. 708–714. DOI: 10.1109/SANER50967.2021.00090 (cit. on pp. 24, 25).

[Pil16] M. Pilkington. "Blockchain technology: principles and applications". In: 2016. DOI: 10.4337/9781784717766.00019 (cit. on p. 23).

[PR19] P. Pal, S. Ruj. "BlockV: A Blockchain Enabled Peer-Peer Ride Sharing Service". In: 2019, pp. 463–468. DOI: 10.1109/Blockchain.2019.00070 (cit. on pp. 33, 35).

[RGA+22] J. Riel, K. Gothe, F. Albrecht, R. Jäger, P. Köhler, A. M. Kunz, L. Matzdorff, T. Reuber, J. Wachsmann, S. Wammetsberger. *AutoRICH: Autonomes Fahren - Risiko & Chancen für die Städte (Endbericht)*. 2022. DOI: 10.5281/ZENODO.8003249 (cit. on p. 17).

[RMP+22] G. S. Ramachandran, S. Malik, S. Pal, A. Dorri, V. Dedeoglu, S. Kanhere, R. Jurdak. "Blockchain in Supply Chain: Opportunities and Design Considerations". In: *Handbook on Blockchain*. Ed. by D. A. Tran, M. T. Thai, B. Krishnamachari. Cham: Springer International Publishing, 2022, pp. 541–576. ISBN: 978-3-031-07535-3. DOI: 10.1007/978-3-031-07535-3_17. URL: https://doi.org/10.1007/978-3-031-07535-3_17 (cit. on pp. 26, 27).

[Sha18] S. Shaheen. "Shared Mobility: The Potential of Ridehailing and Pooling". In: 2018, pp. 55–76. DOI: 10.5822/978-1-61091-906-73 (cit. on p. 22).

[SHY21] S. Singh, A. S. M. S. Hosen, B. Yoon. "Blockchain Security Attacks, Challenges, and Solutions for the Future Distributed IoT Network". In: *IEEE Access* 9 (2021), pp. 13938–13959. DOI: 10.1109/ACCESS.2021.3051602 (cit. on p. 25).

[SKK+22] S. Shirodkar, K. Kulkarni, R. Khanjode, S. Kohle, P. Deshmukh, P. Patil. "Layer 2 Solutions to Improve the Scalability of Blockchain". In: *2022 5th International Conference on Advances in Science and Technology (ICAST)*. 2022, pp. 54–57. DOI: 10.1109/ICAST55766.2022.10039486 (cit. on p. 29).

[SKMM23] T. Stamadianos, N. A. Kyriakakis, M. Marinaki, Y. Marinakis. "Routing Problems with Electric and Autonomous Vehicles: Review and Potential for Future Research". In: *Operations Research Forum* 4.2 (2023). DOI: 10.1007/s43069-023-00228-1 (cit. on pp. 21, 22).

[SMD16] D. Sánchez, S. Martínez, J. Domingo-Ferrer. "Co-utile P2P ridesharing via decentralization and reputation management". In: *Transportation Research Part C: Emerging Technologies* 73 (2016), pp. 147–166. ISSN: 0968090X. DOI: 10.1016/j.trc.2016.10.017 (cit. on pp. 33, 36).

[Sol23] Solidity. 2023. URL: https://soliditylang.org (visited on 09/23/2023) (cit. on p. 60).

[SRF+21] R. Shivers, M. A. Rahman, M. J. H. Faruk, H. Shahriar, A. Cuzzocrea, V. Clincy. "Ride-Hailing for Autonomous Vehicles: Hyperledger Fabric-Based Secure and Decentralize Blockchain Platform". In: 2021, pp. 5450–5459. DOI: 10.1109/BigData52589.2021.9671379 (cit. on pp. 32, 34).

[TK22] D. A. Tran, B. Krishnamachari. "Blockchain in a Nutshell". In: *Handbook on Blockchain*. Ed. by D. A. Tran, M. T. Thai, B. Krishnamachari. Vol. 194. Springer Optimization and Its Applications. Cham: Springer International Publishing, 2022, pp. 3–54. ISBN: 978-3-031-07534-6. DOI: 10.1007/978-3-031-07535-3{\textunderscore}1 (cit. on pp. 23, 25, 28).

[TSA22] M. Tahir, M. Sardaraz, U. Aziz. "Vol 4 Issue 5". In: *International Journal of Innovations in Science and Technology* 4.5 (2022), pp. 52–64. ISSN: 2618-1630. DOI: 10.33411/IJIST/2022040505 (cit. on p. 24).

[Ube23] Uber. 2023. URL: https://www.uber.com/au/en/ride/uberpool/ (visited on 09/23/2023) (cit. on pp. 39, 48).

[UX23] O. C. Uchani Gutierrez, G. Xu. "Blockchain and Smart Contracts to Secure Property Transactions in Smart Cities". In: *Applied Sciences* 13.1 (2023), p. 66. DOI: 10.3390/app13010066 (cit. on pp. 24, 25).

[VL21] E. Vazquez, D. Landa-Silva. "Towards Blockchain-based Ride-sharing Systems". In: 2021, pp. 446–452. DOI: 10.5220/0010323204460452 (cit. on pp. 33, 36).

[vSN+09] J. vom Brocke, A. Simons, B. Niehaves, K. Riemer, R. Plattfaut, A. Cleven. "Reconstructing the Giant: On the Importance of Rigour in Documenting the Literature Search Process". In: 2009 (cit. on p. 29).

[Wei22] X. Wei. "Smart Mobile Information Systems on the Key Systems of Blockchain Privacy Protection". In: *Mathematical Problems in Engineering* 2022 (2022), pp. 1–8. ISSN: 1024-123X. DOI: 10.1155/2022/5126326 (cit. on p. 23).

[WW02] J. Webster, R. T. Watson. "Analyzing the Past to Prepare for the Future: Writing a Literature Review". In: *MIS Quarterly* 26.2 (2002), pp. xiii–xxiii. ISSN: 02767783. URL: http://www.jstor.org/stable/4132319 (cit. on p. 30).

[XCW+22] H. Xiong, M. Chen, C. Wu, Y. Zhao, W. Yi. "Research on Progress of Blockchain Consensus Algorithm: A Review on Recent Progress of Blockchain Consensus Algorithms". In: *Future Internet* 14.2 (2022), p. 47. DOI: 10.3390/fi14020047 (cit. on pp. 23, 24).

[YTH+22] H. Yamanaka, Y. Teranishi, Y. Hayamizu, A. Ooka, K. Matsuzono, R. Li, H. Asaeda. "User-centric In-network Caching Mechanism for Off-chain Storage with Blockchain". In: *ICC 2022 - IEEE International Conference on Communications*. 2022, pp. 1076–1081. DOI: 10.1109/ICC45855.2022.9838289 (cit. on p. 28).

[Zha22] D. Zhao. "Cross-Blockchain Transactions: Systems, Protocols, and Topological Theory". In: *Handbook on Blockchain*. Ed. by D. A. Tran, M. T. Thai, B. Krishnamachari. Cham: Springer International Publishing, 2022, pp. 157–193. ISBN: 978-3-031-07535-3. DOI: 10.1007/978-3-031-07535-3_5. URL: https://doi.org/10.1007/978-3-031-07535-3_5 (cit. on p. 27).

[Zho23] H. Zhou. "Green supply chain financial governance and technology application based on block chain and AI technology". In: *BCP Business & Management* 38 (2023), pp. 881–888. ISSN: 2692-6156. DOI: 10.54691/bcpbm.v38i.3791 (cit. on p. 22).

[ZMM22] H. Zhou, A. Milani Fard, A. Makanju. "The State of Ethereum Smart Contracts Security: Vulnerabilities, Countermeasures, and Tool Support". In: *Journal of Cybersecurity and Privacy* 2.2 (2022), pp. 358–378. DOI: 10.3390/jcp2020019 (cit. on pp. 24, 25).

[ZZHH22] Y. Zhang, Y. Zhou, Y. Hu, H. Huang. "A Decentralized Ride-Hailing Mode Based on Blockchain and Attribute Encryption". In: vol. 13547. 2022, pp. 301–313. DOI: 10.1007/978-3-031-18067-522 (cit. on pp. 32, 35).

All links were last followed on September 25, 2023.

A Smart Contracts

A.1 Smart Contract: Matching.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract MatchingService {

    struct MatchingServiceObject {
        string name;
        uint256 matches;
        uint256 requests;
    }


    MatchingServiceObject[5] public services;

    address public FACTORY_ADDRESS;
    bool public isFactoryAddressSet = false;

    mapping(address => bool) public registeredContracts;
    address[] public registeredContractsList;

    // Declare the event
    event LowestMatchService(string serviceName, uint256 serviceRating);

    modifier onlyFactory() {
        require(msg.sender == FACTORY_ADDRESS, "Only the factory can call this");
        _;
    }

    modifier onlyRegisteredContracts() {
        require(registeredContracts[msg.sender], "Only registered contracts can call this");
        _;
    }

    //Hardcoded Dummy Matching Services
    constructor() {
        services[0] = MatchingServiceObject("ms1", 10, 15);
        services[1] = MatchingServiceObject("ms2", 15, 20);
        services[2] = MatchingServiceObject("ms3", 20, 30);
        services[3] = MatchingServiceObject("ms4", 5, 10);
        services[4] = MatchingServiceObject("ms5", 8, 12);
    }

    function setFactoryAddress(address _factoryAddress) external {
        require(!isFactoryAddressSet, "Factory address is already set");
        FACTORY_ADDRESS = _factoryAddress;
        isFactoryAddressSet = true;
```

```
    }

    function registerContract(address contractAddress) external onlyFactory {
        require(!registeredContracts[contractAddress], "Contract is already registered"); // Additional check to prevent duplicate addresses
        registeredContracts[contractAddress] = true;
        registeredContractsList.push(contractAddress);
    }

    function getAllRegisteredContracts() external view returns (address[] memory) {
        return registeredContractsList;
    }

    function getMatchingService(string[] memory names) public {
        uint256 lowestMatches = type(uint256).max;

        string memory lowestMatchServiceName = "";
        uint256 lowestMatchServiceRating;

        for (uint i = 0; i < names.length; i++) {
            for (uint j = 0; j < services.length; j++) {
                if (keccak256(bytes(services[j].name)) == keccak256(bytes(names[i]))) {
                    if (services[j].matches < lowestMatches) {
                        lowestMatches = services[j].matches;
                        lowestMatchServiceName = services[j].name;
                        lowestMatchServiceRating = (services[j].matches * 100) / services[j].requests; // Multiply by 100 for two decimal places
                        services[j].requests += 1;
                    }
                }
            }
        }
        // Emit the event with the result
        emit LowestMatchService(lowestMatchServiceName, lowestMatchServiceRating);
    }

    function addMatch(string memory serviceName) external onlyRegisteredContracts {
        for (uint i = 0; i < services.length; i++) {
            if (keccak256(bytes(services[i].name)) == keccak256(bytes(serviceName))) {
                services[i].matches += 1;
            }
        }
    }
}
```

A.2 Smart Contract: ContractFactory.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "./contract.sol";
import "./matching.sol";  // Assuming both contracts are in the same directory

contract ContractFactory {

```

```
9       MatchingService private matchingServiceInstance;
10      address[] public registeredContracts;
11
12      uint256 public contractCounter = 0;  // Counter to keep track of contract IDs
13
14      // Mapping from contract ID to contract address
15      mapping(uint256 => address) public contractsByID;
16
17      // Mapping from contract ID to contract timestamp
18      mapping(uint256 => uint256) public timestampByID;
19
20
21      constructor(address _matchingServiceAddress) {
22          matchingServiceInstance = MatchingService(_matchingServiceAddress);
23
24          // Set this contract as the factory address in the MatchingService contract
25          matchingServiceInstance.setFactoryAddress(address(this));
26      }
27
28      mapping(address => Contract[]) public userContracts;
29      event ContractCreated(address indexed user, Contract newContract, uint256 contractID);  // Added
    contractID to the event
30
31      function registerNewContract(address _contractAddress) external {
32          // Call the registerContract() function on the MatchingService contract
33          matchingServiceInstance.registerContract(_contractAddress);
34
35          // Optionally, store the registered contract's address in this factory for record-keeping
36          registeredContracts.push(_contractAddress);
37      }
38
39      function createContract(uint256 _amount) public payable {
40          require(msg.value == _amount, "Sent value does not match the specified amount.");
41          Contract newContract = new Contract{value: _amount}(msg.sender);
42          userContracts[msg.sender].push(newContract);
43
44          // Increment contract counter and map new contract's address to the counter
45          contractCounter++;
46          contractsByID[contractCounter] = address(newContract);
47
48          // Store the current block's timestamp
49          timestampByID[contractCounter] = block.timestamp;
50
51          // Call registerNewContract with the new contract's address
52          this.registerNewContract(address(newContract));
53
54          emit ContractCreated(msg.sender, newContract, contractCounter);
55      }
56
57      function getContractsByUser(address user) public view returns (Contract[] memory) {
58          return userContracts[user];
59      }
60
61      function getContractByID(uint256 contractID) public view returns (address) {
62          return contractsByID[contractID];
63      }
64
```

```
    // Fetch the timestamp by contract ID
    function getContractTimestampByID(uint256 contractID) public view returns (uint256) {
        return timestampByID[contractID];
    }
}
```

A.3 Smart Contract: Contract.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

interface IMatchingService {
    function addMatch(string memory serviceName) external;
}

contract Contract {
    address public party1;
    address public party2;
    bool public isActive;
    bool public rideProviderAcceptedStatus;
    bool public rideProviderArrivedAtPickupLocation;
    bool public userReadyToStartRide;
    bool public rideProviderStartedRide;
    bool public rideProviderArrivedAtDropoffLocation;
    bool public userMarkedRideComplete;
    bool public userCanceldRide;
    bool public rideProviderCanceldRide;

    uint public userRating;
    uint public rideRating;
    bool public isUserRatingSet;
    bool public isRideRatingSet;

    //Hard Coded Address of the Matching Service to bump up Matching Count of Rating Service
    address constant MATCHING_SERVICE_ADDRESS = 0x0991df810C73d820c776b024Eb720d39e9CfBb1a;


    constructor(address _party1) payable {
        party1 = _party1;
        rideProviderAcceptedStatus = false;
        rideProviderArrivedAtPickupLocation = false;
        userReadyToStartRide = false;
        rideProviderStartedRide = false;
        rideProviderArrivedAtDropoffLocation = false;
        userMarkedRideComplete = false;
        userCanceldRide = false;
        rideProviderCanceldRide = false;

    }

    struct Passenger {
        string passengerID;
        uint seatingPosition;
        string startTime;
        uint rating;
```

```
48      }
49
50
51      Passenger[] public passengers;
52
53      function addPassenger(string memory _passengerID, uint _seatingPosition, string memory _startTime)
    public {
54          require(isActive, "Contract is not active.");
55          require(msg.sender == party2, "Only Party2 can add passengers.");
56
57          Passenger memory newPassenger = Passenger({
58              passengerID: _passengerID,
59              seatingPosition: _seatingPosition,
60              startTime: _startTime,
61              rating: 0
62          });
63
64          passengers.push(newPassenger);
65      }
66
67      function addPassengerRating(uint _passengerIndex, uint _rating) public {
68          require(isActive, "Contract is not active.");
69          require(msg.sender == party1, "Only Party1 can rate passengers.");
70          require(_rating >= 0 && _rating <= 5, "Rating must be between 0 and 5.");
71          require(_passengerIndex < passengers.length, "Passenger not found.");
72
73          passengers[_passengerIndex].rating = _rating;
74      }
75
76      function signContract() public payable {
77          require(party2 == address(0), "Party2 has already signed the contract.");
78          require(!isActive, "Contract is already active.");
79          require(!userCanceldRide, "User cannceld ride ");
80          require(msg.sender != party1, "Party2 cannot be identical to Party1.");
81
82          party2 = msg.sender;
83          isActive = true;
84
85          uint256 tenPercent = (address(this).balance * 10) / 100;
86          require(msg.value >= tenPercent, "Party2 must deposit an amount equal to 10% of the contract
    balance.");
87
88          // Refund any excess amount deposited by party2
89          if (msg.value > tenPercent) {
90              payable(msg.sender).transfer(msg.value - tenPercent);
91          }
92      }
93
94      event UpdatePosted(address indexed author, string message, string functionName);
95
96
97      function setRideProviderAcceptedStatus(string memory _message) public {
98          require(isActive, "Contract is not active.");
99          require(msg.sender == party2, "Only Party2 can set the ride provider accepted status.");
100         require(!rideProviderAcceptedStatus, "Ride Provider Accepted Status can only be set once.");
101
102         require(!rideProviderCanceldRide, "Ride Provider Canceld Ride Status can only be set once.");
```

```
103         require(!userCanceldRide, "User Canceld Ride Status can only be set once.");
104 
105         rideProviderAcceptedStatus = true;
106         emit UpdatePosted(msg.sender, _message, "rideProviderAcceptedStatus");
107     }
108 
109     function setRideProviderArrivedAtPickupLocation(string memory _message) public {
110         require(isActive, "Contract is not active.");
111         require(msg.sender == party2, "Only Party2 can set the ride provider arrived status.");
112         require(rideProviderAcceptedStatus, "Ride Provider Accepted Status must be set before setting
    arrived status.");
113         require(!rideProviderArrivedAtPickupLocation, "Ride Provider Arrived Status can only be set once.
    ");
114 
115         require(!rideProviderCanceldRide, "Ride Provider Canceld Ride Status can only be set once.");
116         require(!userCanceldRide, "User Canceld Ride Status can only be set once.");
117 
118         rideProviderArrivedAtPickupLocation = true;
119         emit UpdatePosted(msg.sender, _message, "rideProviderArrivedAtPickupLocation");
120     }
121 
122     function setUserReadyToStartRide(string memory _message) public {
123         require(isActive, "Contract is not active.");
124         require(msg.sender == party1, "Only Party1 can set the user ready to start ride status.");
125         require(rideProviderArrivedAtPickupLocation, "Ride Provider Arrived Status must be set before
    setting user ready to start ride status.");
126         require(!userReadyToStartRide, "User Ready To Start Ride Status can only be set once.");
127 
128         require(!rideProviderCanceldRide, "Ride Provider Canceld Ride Status can only be set once.");
129         require(!userCanceldRide, "User Canceld Ride Status can only be set once.");
130 
131         userReadyToStartRide = true;
132         emit UpdatePosted(msg.sender, _message, "userReadyToStartRide");
133     }
134 
135     function setRideProviderStartedRide(string memory _message) public {
136         require(isActive, "Contract is not active.");
137         require(msg.sender == party2, "Only Party2 can set the ride provider started ride status.");
138         require(userReadyToStartRide, "User Ready To Start Ride Status must be set before setting ride
    provider started ride status.");
139         require(!rideProviderStartedRide, "Ride Provider Started Ride Status can only be set once.");
140 
141         require(!rideProviderCanceldRide, "Ride Provider Canceld Ride Status can only be set once.");
142         require(!userCanceldRide, "User Canceld Ride Status can only be set once.");
143 
144         rideProviderStartedRide = true;
145         emit UpdatePosted(msg.sender, _message, "rideProviderStartedRide");
146     }
147 
148     function setRideProviderArrivedAtDropoffLocation(string memory _message) public {
149         require(isActive, "Contract is not active.");
150         require(msg.sender == party2, "Only Party2 can set the ride provider arrived at dropoff location
    status.");
151         require(rideProviderStartedRide, "Ride Provider Started Ride Status must be set before setting
    ride provider arrived at dropoff location status.");
152         require(!rideProviderArrivedAtDropoffLocation, "Ride Provider Arrived At Dropoff Location Status
    can only be set once.");
```

```
153
154         require(!rideProviderCanceldRide, "Ride Provider Canceld Ride Status can only be set once.");
155         require(!userCanceldRide, "User Canceld Ride Status can only be set once.");
156
157         rideProviderArrivedAtDropoffLocation = true;
158         emit UpdatePosted(msg.sender, _message, "rideProviderArrivedAtDropoffLocation");
159     }
160
161     function setUserMarkedRideComplete(string memory _message) public {
162         require(isActive, "Contract is not active.");
163         require(msg.sender == party1, "Only Party1 can set the user marked ride complete status.");
164         require(rideProviderArrivedAtDropoffLocation, "Ride Provider Arrived At Dropoff Location Status
    must be set before setting user marked ride complete status.");
165         require(!userMarkedRideComplete, "User Marked Ride Complete Status can only be set once.");
166
167         require(!rideProviderCanceldRide, "Ride Provider Canceld Ride Status can only be set once.");
168         require(!userCanceldRide, "User Canceld Ride Status can only be set once.");
169
170         userMarkedRideComplete = true;
171         //Call Matching Service, ms1 is hardcoded. For a real implementation this value would be provided
     by the forntend when calling the setUserMarkedRideComplete() function
172         IMatchingService(MATCHING_SERVICE_ADDRESS).addMatch("ms1");
173         emit UpdatePosted(msg.sender, _message, "userMarkedRideComplete");
174     }
175
176     function setUserCanceldRide(string memory _message) public {
177         require(msg.sender == party1, "Only Party1 can set the user canceld ride status.");
178
179         if(!isActive) {
180             uint256 balance = address(this).balance;
181             payable(party1).transfer(balance);
182             return;
183         }
184
185         require(!rideProviderCanceldRide, "Ride Provider Canceld Ride Status can only be set once.");
186         require(!userCanceldRide, "User Canceld Ride Status can only be set once.");
187
188         userCanceldRide = true;
189
190         if(isActive) {
191             uint256 balance = address(this).balance;
192             payable(party2).transfer(balance);
193         }
194
195         emit UpdatePosted(msg.sender, _message, "userCanceldRide");
196     }
197
198     function setRideProviderCanceldRide(string memory _message) public {
199         require(isActive, "Contract is not active.");
200         require(msg.sender == party2, "Only Party2 can set the ride provider canceld ride status.");
201
202         require(!rideProviderCanceldRide, "Ride Provider Canceld Ride Status can only be set once.");
203         require(!userCanceldRide, "User Canceld Ride Status can only be set once.");
204
205         rideProviderCanceldRide = true;
206
207         uint256 balance = address(this).balance;
```

```
        payable(party1).transfer(balance);

        emit UpdatePosted(msg.sender, _message, "rideProviderCanceldRide");
    }

    function setUserRating(uint _rating) public {
        require(msg.sender == party2, "Only Party2 can set the user rating.");
        require(!isUserRatingSet, "User rating can only be set once.");
        require(isActive, "Contract is not active.");
        require(_rating >= 0 && _rating <= 5, "Rating must be between 0 and 5.");
        userRating = _rating;
        isUserRatingSet = true;
    }

    function setRideRating(uint _rating) public {
        require(msg.sender == party1, "Only Party1 can set the ride rating.");
        require(!isRideRatingSet, "Ride rating can only be set once.");
        require(_rating >= 0 && _rating <= 5, "Rating must be between 0 and 5.");
        require(isActive, "Contract is not active.");
        rideRating = _rating;
        isRideRatingSet = true;
    }


    function claimETH(uint256 amount) public {
        require(isActive, "Contract is not active.");
        require(msg.sender == party2, "Only Party2 can claim the deposited ETH.");
        require(userMarkedRideComplete, "User must mark the ride complete before claiming the deposited
ETH.");
        require(amount <= address(this).balance, "Requested amount exceeds the contract balance.");

        address payable hardcodedAddress = payable(0xE39a3085CB78341547F30a1C6bD12977d51aa967);  //
replace with the actual GETACAR Foundation address

        uint256 balance = address(this).balance;
        uint256 tenPercent = balance / 10;
        uint256 remainder = balance - tenPercent;

        hardcodedAddress.transfer(tenPercent);

        uint256 payback = remainder - amount;
        remainder -= payback;

        payable(party1).transfer(payback);
        payable(party2).transfer(remainder);
    }

}
```

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature