Cryptocurrency as a Base for a Decentralized Application to Improve Security and Cost Reduction in the Ridesharing Industry

Cryptocurrency as a Base for a Decentralized Application to Improve Security and Cost Reduction in the Ridesharing Industry

Alejandro Covarrubias

A Thesis in the Field of Software Engineering

for the Degree of Master of Liberal Arts in Extension Studies

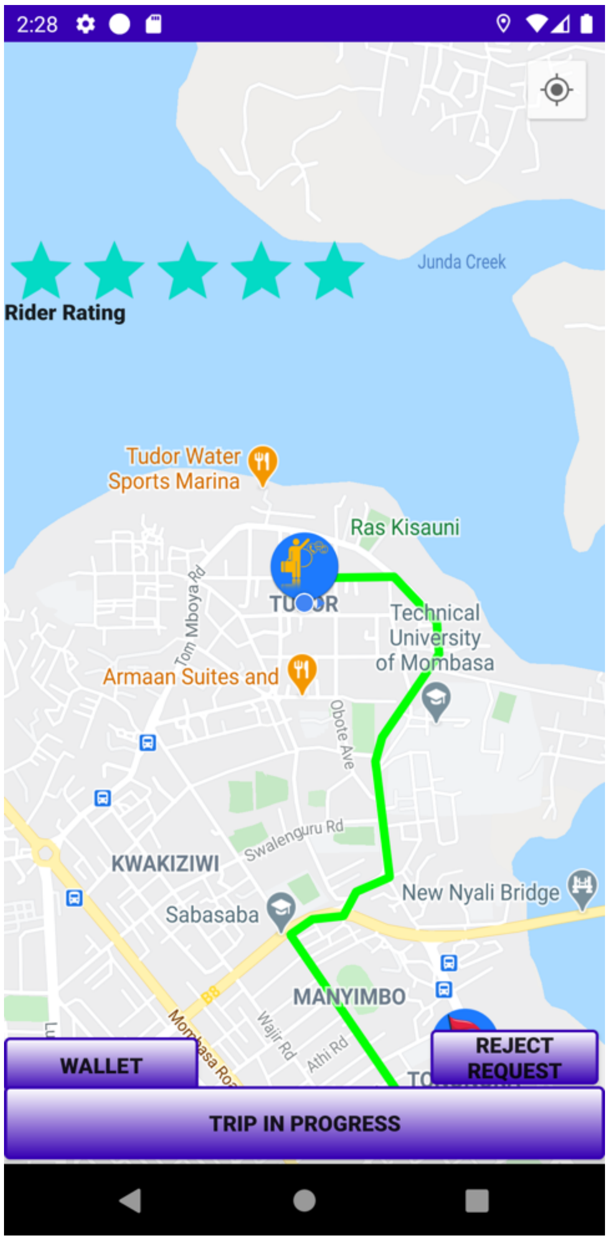
Harvard University

May 2021

Abstract

The thesis introduces a ride-sharing application based on blockchain. The platform is designed to enhance the ride-sharing industry’s security and improve the driver’s profit, the cost for riders, and payment processing performance in the ride-sharing industry.

Frontispiece



Author’s Biographical Sketch

I was born in 1984 in Hermosillo, Sonora, Mexico, when Apple just launched the Macintosh. My favorite game was playing with Legos during childhood, as building ships, cars, and robots kept my imagination busy. I also enjoyed reading books about science fiction and found a home for my favorite author on the foundation and future portrayed by Isaac Asimov. While in elementary school, I noticed how most of my classmates were not fond of mathematics and science. I particularly enjoyed those subjects because I found them exciting and challenging. As I grew up, my interest in courses that challenged me, such as science and technology, kept increasing. I was in high school. The world of technology had massive changes as computers and information available to everybody through the World Wide Web kept growing as an open-source of knowledge and social communication. I found it very interesting how technology was developing in front of my eyes and decided to pursue a technology career. When I was graduating high school, there were plenty of advances in hardware electronics and software technology. Hence, I decided to study a major that will allow me to learn both hardware and software engineering and work and research both fields.

Enter Electrical Engineering. The major offered the ability to learn electronic circuits. I learned how to build a computer from the barebone motherboard and insight into how the software works from binary code and how hardware can take software code and translate it to executable actions. During my college days, I had exposure to courses that require learning programming basics such as Programming and Problem Solving based in C, Microprocessor Organization based in C, and Fundamental Computer Architecture based in Java. By the time I graduated in 2009, the landscape of technology had changed since the demand for Software Engineering jobs increased on the market while Electrical Engineering jobs kept decreasing.

After my bachelor's graduation, I strived to improve my software development skills. I gain experience working as a Software Automation Engineer on Verizon projects, Software Quality Assurance Engineer on projects for Symantec, and more recently as a Software Engineer for Sony Interactive Entertainment. At Sony Interactive Entertainment, I currently develop software for the PlayStation commerce platform, concentrated on payments and continuous delivery, continuous integration.

With the research performed as a result of the thesis, I aim to provide an implementation that will serve as a platform to improve efficiency in several areas for the ride-sharing industry. The project seeks to support ushering in a new era in payments via an application that can become mainstream in the ride-sharing industry

Acknowledgments

I want to thank Eric Gieseke for agreeing to be my thesis director and for all the ongoing support and guidance throughout this project. His advice throughout the project was invaluable.

Thanks to my research advisor Professor Hongming Wang for reviewing my thesis proposal and thesis and providing invaluable advice.

Thanks to Professor Sylvain Jaume, director and research advisor for his guidance during my studies.

Thanks to my father for always assuring me that through dedication, I can accomplish anything I set my mind to.

Finally, I would like to thank my wife for her patience and support as she always stood beside me through all the coursework and thesis work.

Table of Contents

List of Tables

List of Figures

Chapter I.

Introduction

The thesis project proposes a ride-sharing platform based on blockchain to enhance the ride-sharing industry's security and payment processing performance on the ride-sharing industry.

In current ride-sharing services, the centralization of the authority in the management of data generated by devices lacks transparency in how a device shares user data among third-party entities (Ayoade, 2018). The project aims to clarify data management and enhance the ride-sharing industry's security by introducing a decentralized platform.

Companies such as Uber and Lyft act as intermediaries that charge 20 percent commission for each ride transaction (Hmielewicz, 2018). The proposed ride-sharing platform provides a financial or economic enhancement over the most popular solutions by securing a direct payment without intermediaries.

The overview chapter of the research describes the historical overview of the ride-sharing economy. The overview describes the thesis project's importance on the ride-sharing market, the benefits, challenges, and opportunities for improvement on current popular ride-sharing solutions. The thesis then dives into the details of the technologies researched and considered proposing an implementation for the project. The high-level requirements then describe the architecture and design of the ride-sharing platform. Finally, the thesis covers the ride-sharing platform's deployment and testing, a summary of results, and future work for the ride-sharing platform.

Chapter II.

Overview

This chapter describes an overview of the research. The first section of the overview introduces the essential facts regarding the historical overview of the ride-sharing industry. The following sections will then dive into ride-sharing and describe the ride-sharing market, the benefits, challenges, and target areas for improvements.

2.1. Historical Overview

Ridesharing services, sometimes referred to as ride-hailing services, by definition arrange shared rides on short notice. The main difference between ridesharing and a taxi is that passengers need to specify their pickup location and destination with ride-sharing before hailing a ride. Selecting pickup location and destination allows ride-sharing services to provide the passenger with a price estimate for the ride. (Dalul, 2019).

The history of ridesharing in America dates back to World War I. During the World War I period, the United States of America government required people to utilize ridesharing to conserve rubber for the war. Later on in the 1970s, the oil crisis and spikes in gas prices increased ridesharing interest. From then on, ridesharing has existed in numerous forms, such as carpool, vanpool, or airport shuttle (Hahn, 2017).

In 2009, Uber was founded in San Francisco to challenge what was generally considered the city's inefficient and inadequate taxi service. Since then, Uber has rapidly spread its services worldwide, and by 2017 it operates in more than seventy countries, with around $16 billion invested in the company since its inception. The ride-hailing company has achieved extremely rapid global expansion through outmaneuvering governments, regulators, and competitors. Uber based its rise on a deliberate strategy of acting as a market disruptive innovator through a user-friendly technology and using the sharing economy (Dudley, 2017).

Companies such as Uber or Lyft provide a ride-sharing application that allows customers to access rides with the tap of a finger. Uber has diversified its business into food delivery, bikes, and scooters. Uber's net revenue is 2.95 billion dollars and grows 38 percent over one year (Hmielewicz, 2018). However, widespread, current mainstream ride-share solutions have security risks for passengers and drivers (Thomson, 2016).

2.2. Package and Food Delivery

The current outbreak of Covid-19 has forced organizations worldwide to shift the trajectory in which businesses work and redefine office space dynamics. As of date, it is perhaps the most significant social experiment of the future work in action, with every organization redefining their work from home and social distancing policies. However, the impact on employment is far more profound than just changing where people work. It is also fundamentally altering the work that employees need to perform and accomplish (Jesuthasan, 2020).

Companies in China started creatively sharing employees and moving them from low demand businesses like restaurants and moving them to others with a demand like Hema. Hema is Alibaba's retail grocery chain known for its fast grocery food delivery. More than 3,000 new employees from more than 40 companies in different sectors joined Hema’s employee sharing plan (Jesuthasan, 2020).

Recent research shows that the rise of food delivery has become a significant trend among customers across all demographics. In 2018, Frost and Sullivan estimated that the food delivery service industry generated 82 billion in gross revenue and predicted that this number would more than double by 2025 (McCarthy, 2020).

Enter the need for a platform that can provide the features for delivering packages such as food or any other goods as a service. The platform proposed as a result of this research would support the ride-sharing of passengers. Most importantly, the platform will also support the feature of delivering packages as a service. Markets shifting to use or accept cryptocurrencies such as Kenya will potentially adopt a ride-sharing platform that can provide ride and package delivery services via digital payments.

2.3. Benefits of Ridesharing

Ridesharing has eased congestion on the roads by people sharing cars. Before ridesharing hit the mainstream, governments, and organizations were promoting the benefits of ridesharing.

Decreasing congestion is one way to contribute to saving the environment. If everyone could fill a car of four people, this would mean four times fewer greenhouse gas emissions. Decreasing congestion would significantly change the amount of greenhouse gas in the atmosphere and slow down the rate of warming (Agrawal, 2016).

The rapid growth of ridesharing services, which has caused the taxi industry's contraction and reduced transit's share of passengers, suggests that utility-maximizing travelers have improved their welfare by shifting to a new model. A Hutchins Center research quantified those benefits in San Francisco Bay Area markets, accounting for travelers' complete set of transportation options. The Hutchins Center research found that travelers have gained roughly $1 billion annually from Uber's service (Hwang, 2020).

Hutchin Center research also suggests that after decades of inefficiency and technological stagnation in urban transportation, ridesharing is a welcome innovation followed by other transportation innovations. It would be unwise to discourage entrepreneurs' innovative efforts by protecting the less efficient modes displaced (Hwang, 2020).

2.4. Ridesharing opportunities for improvement

A target area to improve on current ride-sharing solutions is that of security risks. Security risks include passengers facing problems like reckless driving and harassment. Both drivers and passengers are also concerned about the misuse of their data. Previous research also shows that the users are putting their trust in these services and occasionally preferring them over public transportations (Majumder, 2019).   A ride-sharing platform can address these risks by introducing a blockchain-based solution to maintain decentralized and secure user data of drivers and riders (Ayoade, 2018). The blockchain-based solution can dramatically improve security by validating users and drivers before they participate in the system.

Another target area to improve on current ride-sharing solutions is to reduce the cost for riders and enhance the profit of drivers over existing popular ride-sharing solutions by using blockchain technology.

Mainstream ride-sharing platforms act as a third party or middle man that keeps 20 percent of the transaction. (Mire, 2018). One of the possible use cases of Blockchain technology is that of enhancing current ride-sharing solutions. A ride-sharing platform could eliminate the middleman if a platform as a service is built and provided as an alternative option based on blockchain technology as its core foundation. Such a platform will provide drivers and riders a service to facilitate direct and secure payment transactions.

A ride-sharing application based on blockchain technology can improve the driver experience by providing immediate payment at service completion. A direct payment contrasts with credit card payments, which can take a day to transfer money from one party to another (Swan, 2015). It can take days to transfer money when the transaction sender and receiver have accounts in different countries. The global payments business is a large, slow, costly, and error-prone industry (Swan, 2015). According to the American Bankers Association, by the end of 2017, there were 364 million active credit card accounts. If you're an owner of one of those credit cards, then you may have had an erroneous charge on your account that you needed to dispute at some point. When consumers report charge disputes to their credit card issuer, the process of a chargeback begins (Schaut, 2020). Chargebacks are generally very bad for merchants as they often come with fees that range between $20 and $100. If a business has too many chargebacks as a percentage of its total transactions, the business account is prone to deactivation, or its per-transaction costs may significantly increase (Yowana, 2020).

The payment of a given transaction can be improved using a blockchain-based payment processor, which has the potential to add tremendous value in this space by reducing the multi-day payment cycle down to real-time (Swan, 2015). The ride-share application based on a blockchain-based payment processor will also remove the average 20 percent commission currently charged on most popular ride-share solutions and drastically improve drivers' profit.

Another opportunity is providing a decentralized authority in managing how a device shares data in general (that being, user data, transactions, or smart contracts). A decentralized authority can address the problem with the lack of transparency in how a device shares user data among third-party entities in the current popular ride-sharing solution.

Chapter III.

Important Technologies

The chapter dives into the details for each of the crucial technologies considered to research and implement a solution that will address the critical improvements targeted over current popular ride-sharing solutions.

3.1. Cryptocurrency

A cryptocurrency is a digital or virtual currency secured by cryptography, making it nearly impossible to counterfeit. Cryptocurrencies are decentralized networks based on blockchain technology (Frankenfield, 2020). A cryptocurrency-based platform enables a secure and direct payment transaction, reducing the cost of transactions by eliminating intermediaries. It also minimizes the time required to complete a payment transaction by removing banks or other institutions that might slow down the payment processing process.

Interest in virtual currencies is rising across the African continent, especially in Kenya, the so-called African silicon valley. Despite warning from Kenya's central bank about cryptocurrencies' volatility, some Nairobi businesses are now accepting Bitcoin payments. According to the Blockchain Association of Kenya, the total number of Bitcoin transactions in Kenya is worth over $1.5 million (Russon, 2019).

Stable coins are a popular type of digital currency available on blockchains. Stable coins tie their value to a fiat currency. For example, the USD Coin (USDC), set their value in relation to the US dollar since it is the most widely used fiat currency in the traditional international market. Due to their digital nature and transactional simplicity, these assets have become an attractive alternative to make cross-border transactions and remittances across North American and European markets. However, their increasing adoption as a store of value in emerging markets has set a viable alternative for payment transactions (Serrano, 2020).

The implementation of this project proposes the use of cryptocurrency as the solution to pay for ridesharing. Digital currencies are a new way to store an easily transferable value anywhere without involving institutions or involving the traditional banking solutions while also providing a safeguard against the devaluations previously mentioned. The current economic scenario opens up the door for digital currencies to play the role as an alternative to secure financial assets in the given monetary crisis.

3.2. Smart Contract

A smart contract is a critical component of many platforms and applications using blockchain or distributed ledger technology. “Smart contracts” is a term used to describe computer code that automatically executes all or parts of an agreement and stored on a blockchain-based platform (Levi, 2018).

Imagine a legal contract where an application takes specific actions under conditions. Even after the parties sign such an arrangement, they must rely on the other’s good faith to carry out their side of the agreement (Bambara, 2018). A given application can address the good faith risk using a smart contract based on a blockchain (Bambara, 2018). The two parties can collectively write conditions of their agreement in computer code and initialize a smart contract to secure enough funds for both parties to have no risks. A smart contract based on blockchain technology cannot be tampered with or modified, and they are automatically executed by computer code, hence, eliminating the good faith element.

3.3. Internet of Things

The Internet of things is connecting any device with an on and off switch to the Internet. The devices can range from cell phones, coffee makers, washing machines, headphones, lamps, and wearable devices. IoT also applies to machines' components, such as a jet engine of an airplane or an oil rig drill. The analyst firm Gartner says that by 2020 there will be over 26 billion connected devices. The IoT is a giant network of connected "things" (which also includes people). The relationship will be between people-people, people-things, and things-things.

3.4. Decentralized Management with Blockchain

The blockchain verifies each transaction in the public ledger by the consensus of a majority of the participants in the system. And, once entered, nobody can erase the information from the blockchain. The blockchain contains a verifiable record of every single transaction ever made on the blockchain (Memon, 2018).

Blockchain technology provides decentralized management of assets, such as decentralized currency management, as seen in Bitcoin. Blockchain's proponents expect the blockchain to be virtually immutable without being centralized. In other words, the blockchain does not require a trusted third party that decides upon the ledger's content. Bitcoin, the first blockchain implementation, has succeeded in allowing for digital payments without having to rely on any trusted third party. Decentralization can bring cost savings (through disintermediation) and empowerment to participants. The blockchain parties do not need to trust a powerful third party to act in their best interest (Halaburda, 2019).

The platform implemented as an application resulting from this research proposes a decentralized data management system for smartphone devices where the smart contracts enforce all data access permissions. The platform stores the audit trail of data access in the blockchain.

3.5. Service Oriented Architecture

A monolith is a single unit that contains many inter-dependencies. Hence, one bug that might affect a tiny part of a single feature can cause problems in an unrelated area or even bring an entire application down. Consequently, a possible chain of errors could occur, and therefore, it becomes difficult to isolate the root cause of any bug that might emerge. Knowing this, developers must thoroughly test each module, as it may be impossible to predict the outcome of any change to the monolith codebase (Wittmer, 2020).

Enter Service Oriented Architecture (SOA) is an architecture that allows services to communicate across different platforms and languages by implementing what is known as a “loosely coupled” system. While the concept of SOA has been around for many years, it is only within the past decade that it has risen to the forefront of software-related technologies.

The term “loose coupling” refers to the client of a service and its ability to remain independent of the service that it requires. The most important part of this concept is that the client, which in itself can be, a service, can communicate with the service even if they are not closely related. The service achieves communication by implementing a specified interface that can perform the necessary actions to transmit data (Watts, 2017).

3.6. Microservices

A microservice architectural style is a refinement of SOA. It is an approach to developing a single application as a suite of small services, running in its process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use other data storage technologies (Lewis, 2014).

Microservices do not have the same resilience, fault tolerance, and isolation issues that plague monolith applications. Microservices are loosely coupled; hence, there is little chance that a bug found in one microservice module will compromise other services in the application. That separation between microservices makes it easy to isolate and fix problems while creating less customer impact (Wittmer, 2020).

3.7. Algorand Platform

Another significance of this research was to provide a viable solution to current blockchain scalability issues. Bitcoin uses blockchain to be more secure, but so far, the implementation has encountered scalability issues (Yli-Huumo, 2016). One problem is that with each transaction, we have a maximum size of 1MB (a Bitcoin block now supports 32 MB) per block, and there are only so many payments that Bitcoin can process at once. Currently, at a maximum, Bitcoin can handle about sixty transactions per second

(Yli-Huumo, 2016). As a result of this research, the implementation will address the scalability issue by leveraging the use of the Algorand platform.

Algorand is a blockchain platform that aims to create a borderless economy via its decentralized public blockchain. The platform's core follows simplicity, instant transactions, direct usage, adoption, and performance (Walters, 2019).

An essential feature for this project's research and implementation is that the Algorand protocol core design prioritizes speed. In theory, developers have reached optimal efficiency by finalizing Blocks in just one round of voting. The process to finish blocks ensures Algorand protocol completes each block instantly and dramatically increases the transactions per second handled by the network. The main net has launched with the capability of handling 1,000 transactions per second with a latency that is fewer than 5 seconds (Walters, 2019).

The original Bitcoin blockchain can reportedly process around seven TPS, which prevents the technology from scaling up and ultimately slows down Bitcoin's opportunity as a mainstream payment solution. At the end of 2017, Bitcoin users had the option to pay $28 US dollars in fees, so their transactions would not take days to complete (O'Neal, 2019). The Centre Consortium recently announced that USDC will now use Algorand. USDC has proliferated, with over 1.8 billion in circulation and customer usage growth three times in the past six months (Alois, 2020). The use of Algorand will facilitate speed and scale up the development and performance of payments and financial applications (Alois, 2020). It has been confirmed in practice that the Algorand platform will add the support of over 1,000 TPS and transaction fees of 1/20th of a cent to the USDC ecosystem, a performance that is far better than Bitcoin.

Chapter IV.

High-Level Rideshare Requirements

This chapter describes the high-level ride-share requirements for the ride-share application. The high-level rideshare requirements cover the functional and non-functional requirements of the application.

The functional requirements will cover the features that allow the ride-sharing platform to function as intended to cover the application's basic needs as a system.

The use case section will then cover the general use case scenarios that the platform must cover to meet the functionalities required to provide a reliable service for the different actors that the service targets. The basic system requirements must cover the mobile application use cases presented in section 7.2 of this chapter.

Finally, the chapter's non-functional requirements section will cover the system behavior, features, and characteristics that impact the ride-sharing application user experience.

4.1. Functional Requirements

A ride-sharing android application must service users' interface to interact with the ride-sharing platform using android devices. The android application must provide the interfaces for both the driver and the users seeking a ride or delivery service. The driver's interface must cover the functionalities to create an account, link a page account, log in to the service, check or verify the wallet balance, search for riders, verify and perform rider rating. The rider's interface must provide the options to create an account, link a Pago account, verify the wallet balance, search for drivers, select a destination, package delivery, preferred pick-up location, verify driver rating, and rate a driver after trip completion. The payment issued to the driver must be a live payment that is received and or signed right after trip completion.

4.2. Use Cases

The use case section covers the high-level functional requirements covered by the ride-sharing platform to meet the functionalities outlined in the Functional Requirements section.

The ride-sharing platform must support the use cases outlined in Figure 1 to cover basic functionalities from a rider's perspective.

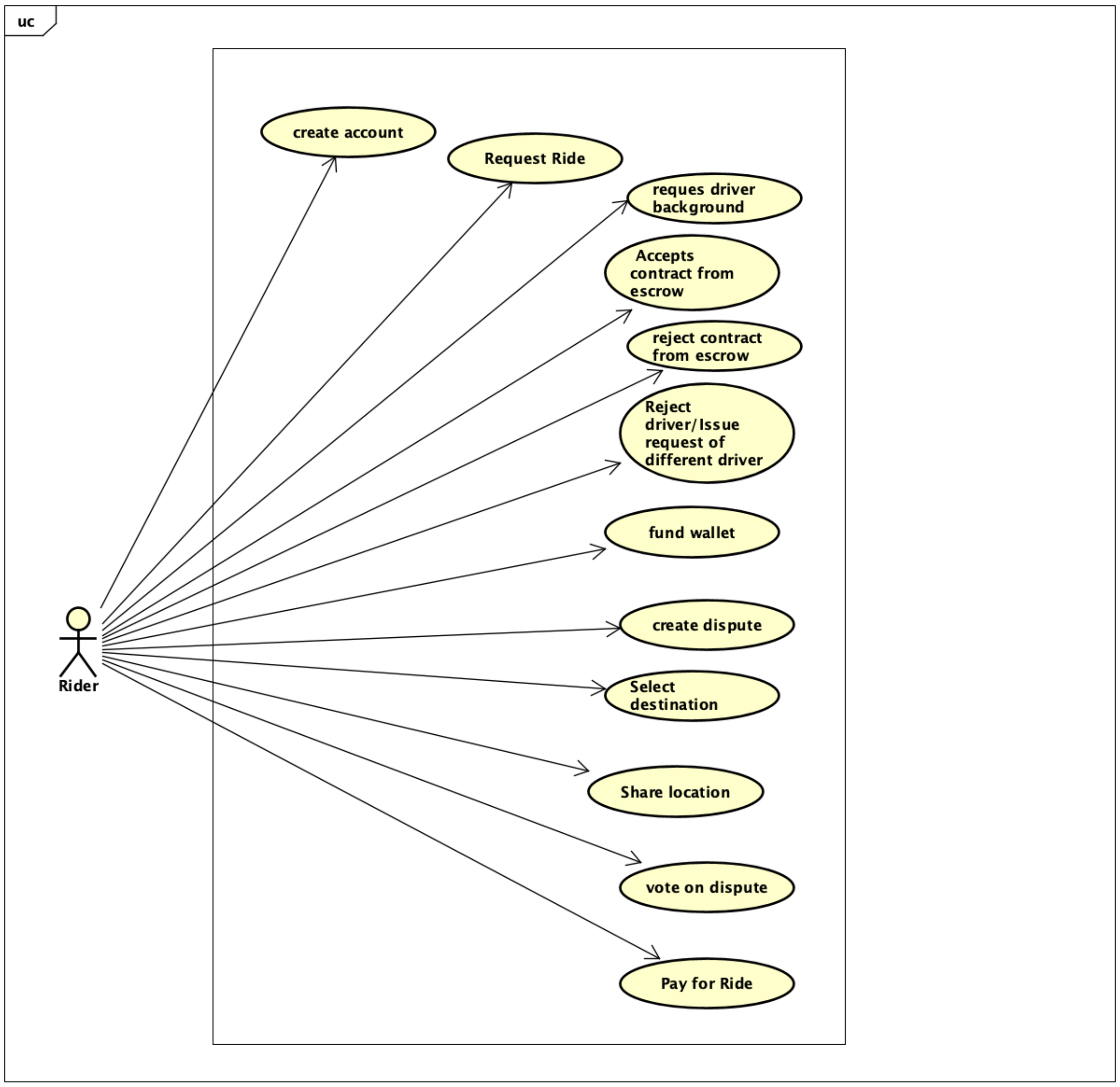


Figure 1. Rider Use Cases

A high-level overview of the prominent rider use cases that the ride-sharing platform must cover.

The ride-sharing platform must support the use cases outlined in Figure 2 below to cover basic functionalities from a driver's perspective.

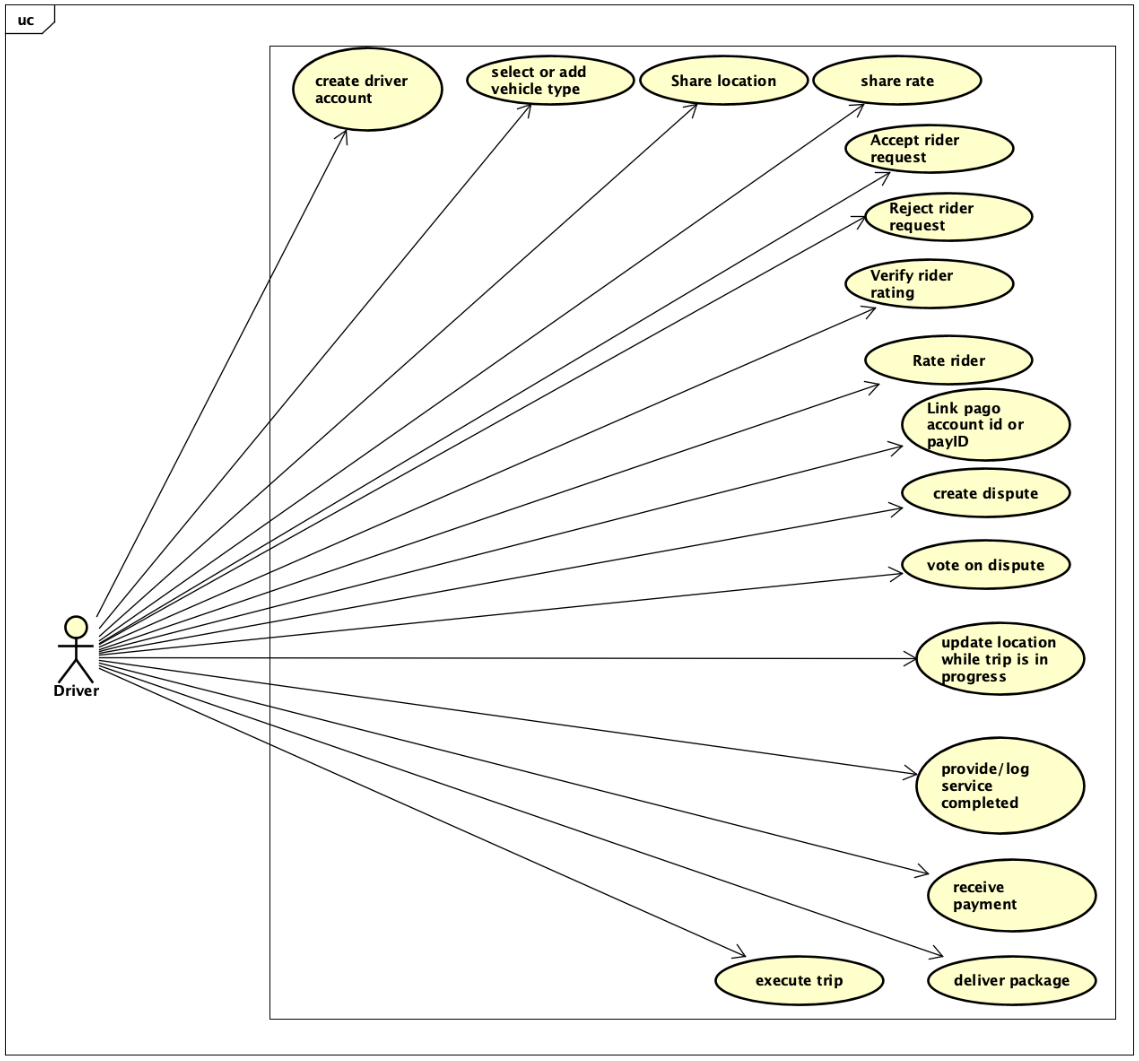


Figure 2. Driver Use Cases

A high-level overview of the prominent driver use cases that the ride-sharing platform must cover.

The ride-sharing platform must support the use cases outlined in Figure 3 below. The given use cases concentrate on scenarios that could arise due to a dispute initiated by the driver or rider.

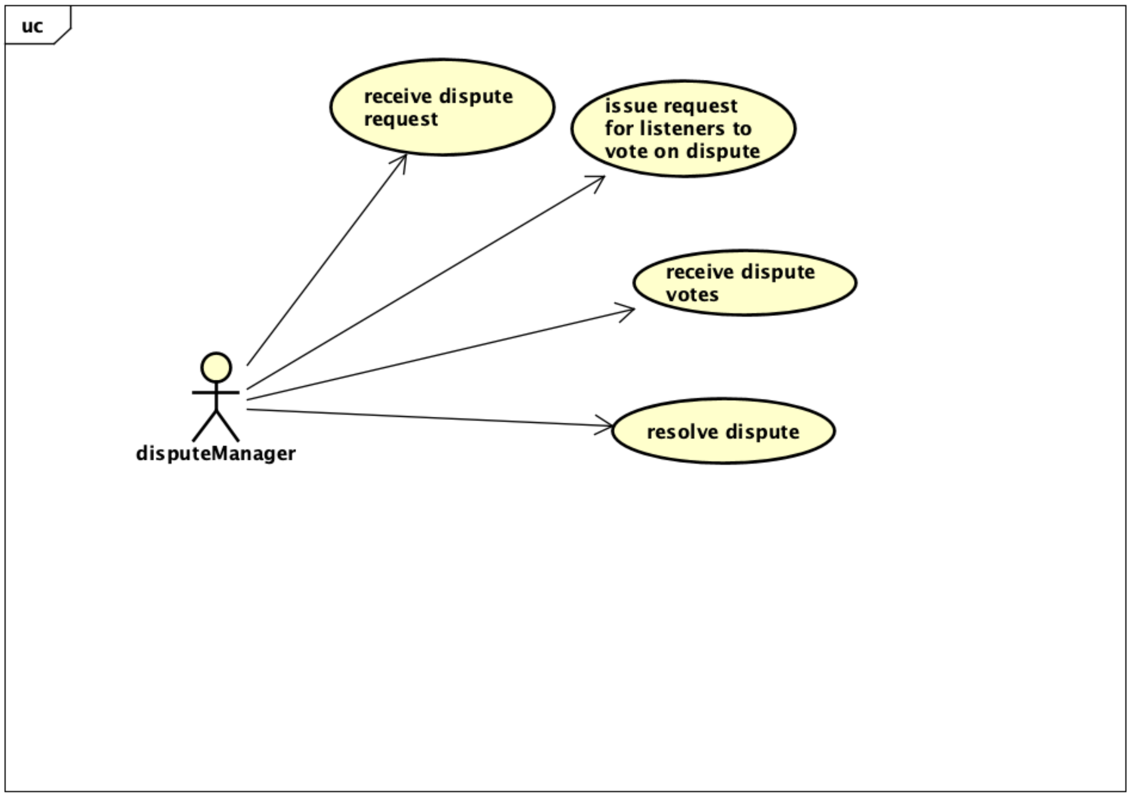


Figure 3. Dispute Use Cases

A high-level overview of the prominent dispute use cases that the ride-sharing platform must cover.

4.3. Non-Functional Requirements

The ride-sharing application must meet the following non-functional requirements. The android ride-sharing application's functional requirements must work regardless of the locale of the driver or rider. In other words, the android ride-sharing application will meet non-functional requirements if the features work as a worldwide application. The platform must follow a microservice architecture to address any scalability and resiliency issues. The microservice architecture infrastructure must account for scaling up if the number of instances to service the requests is insufficient to handle riders and drivers' volume. The ride-sharing application must support finding multiple drivers for the rider to select a driver for a ride request. The number of drivers to be found upon a ride request must be a configurable setting. The android ride-sharing application must control all the decision making logic from the ride-sharing platform without a centralized entity (no groups or organizations will control decision making).

Chapter V.

Architecture

This chapter introduces the high-level architecture and the software technologies required for the architecture of the ride-sharing platform. Figure 4 presents the high-level component diagram that illustrates the platform's main elements. There are three microservices, and each of the microservices has a dedicated Cassandra database. The android ride-sharing application issues API requests to the microservices to meet the requirements to exercise both the rider and the driver's use cases. An external Pago service application is used for the user to create a Pago account and sign transactions.

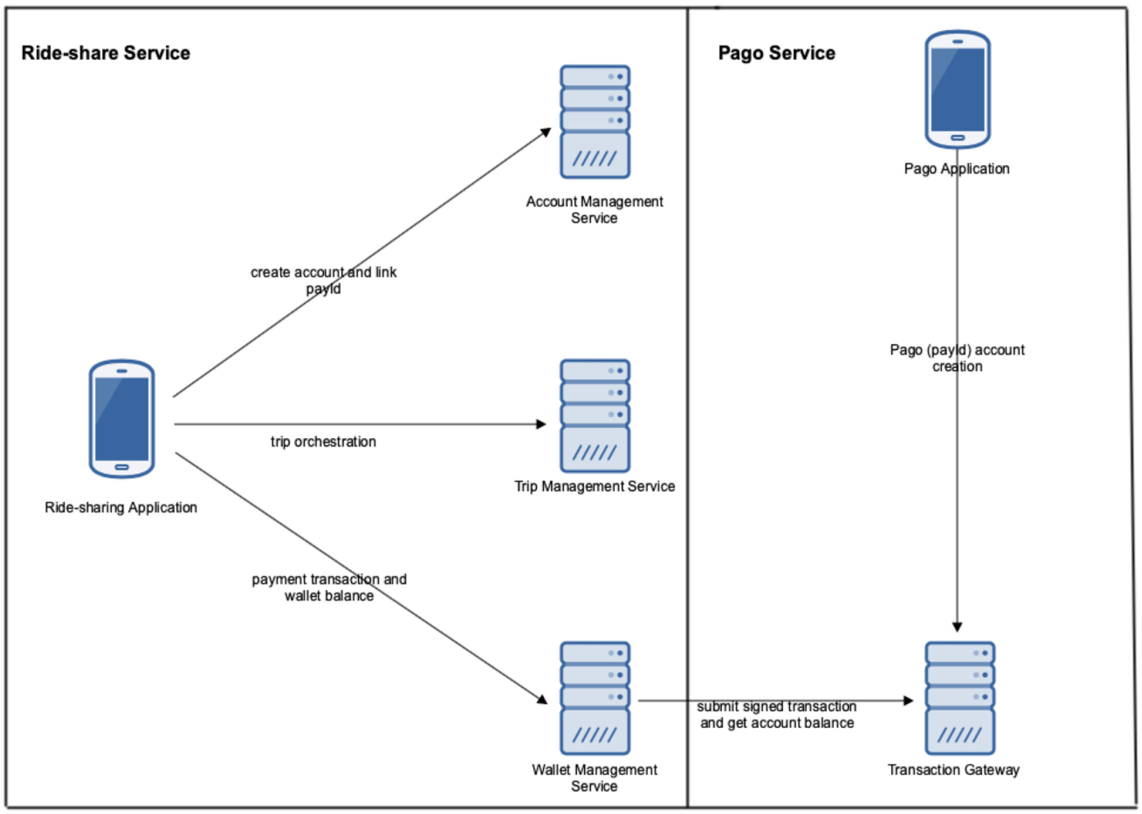

Ride-share Service
Account Management Service
create account and link payId
Ride-sharing Application
trip orchestration
Trip Management Service
payment transaction and wallet balance
Wallet Management Service
Pago Service
Pago Application
Pago (payId) account creation
submit signed transaction and get account balance
Transaction Gateway

Figure 4. Component diagram

High-level component diagram.

The ride-sharing platform will initiate the account management by requests from the android ride-sharing application to service account creation requests, link Pago account, and authentication. The android ride-sharing application creates a REST API request to the account management microservice that will read and write data on the account table on Cassandra's ride-sharing database. The microservice writes the account data to the account table as part of the account creation flow. The microservice reads the account data from the account table as part of the authentication flow.

The trip management service handles the trip orchestration of the ride-sharing platform. Initially, the research proposed had a target to use Cadence as the module for trip orchestration. However, while in the research implementation phase, there were no considerable benefits from using Cadence instead of implementing an orchestrator that followed the command pattern. The risks and future works chapter discuss more details on the advantages and disadvantages of both approaches. The microservice reads and writes the account and location data to the trip table as part of the trip orchestration flow.

The wallet management microservice handles the payment transaction and wallet balance events issued by the ride-sharing android application. The wallet management microservice serves as a gateway to issue calls to Pago's transaction gateway to get the balance of a given Pago account and complete a payment transaction. The wallet management microservice sends unsigned transactions to the Transaction Gateway, and the Transaction Gateway coordinates the interaction between the business application and the Algorand blockchain. The microservice reads and writes the payment transaction data to the wallet table.

5.1. Software Technologies

The purpose of this section is to outline the selected software technologies used to meet the design requirements, as stated in the previous section.

The platform's high-level architecture is microservice-based. The ride-sharing application used the microservice architecture to divide the platform functionalities into microservices based on the business process. The services are interconnected, becoming a single application with a complete business process. One of the architecture's advantages is that a given platform can add more microservices without affecting others (Suryotrisongko, 2017). The ability to add new microservices will provide functional extensibility for the ride-sharing platform.

Spring, a leading development framework, provides a comprehensive set of libraries for microservice architecture. Spring is a standard development framework for building Java-based applications. Spring's dependency injection framework supports managing large java projects by externalizing the relationship between objects through annotations rather than those objects having hard-coded knowledge about each other (Carnell, 2017).

Maven provides for the management of libraries and software dependencies. Maven has a defined lifecycle and standard plugins that know how to build libraries and web applications. Maven requires minimal effort to compile, manage dependencies, and run unit tests from the command line (Sonatype, 2008).

Cadence fault-oblivious stateful code platform preserves a complete multithreaded application state, including thread stacks with local variables across hardware and software failures. It greatly simplifies the coding of complex stateful distributed applications. Simultaneously, it is scalable and robust enough to power hundreds of critical use cases at Uber and outside (Ahmad, 2020). Cadence is the orchestration engine to be used to manage processes for the application trip.

Cassandra database is the technology chosen to be used to manage and store data for the ride-sharing platform. Cassandra is a very scalable and resilient database that is easy to master and simple to configure, providing neat solutions for complex problems. Cassandra's database includes event logging, metrics collection, and monitoring of the historical data. These tasks are quite hard to accomplish correctly, given the variety of operating systems, platforms, browsers, and devices both startup products and enterprise systems face in their daily operations (Fedak, 2018).

Algorand blockchain is the platform chosen to process digital payments. Algorand is a high-performance next-generation blockchain whose goal is to create a transparent system in which everyone can achieve success through decentralized projects and applications. Algorand stands out from other high-performance blockchains by the credibility of its founder, MIT professor Silvio Micali. Professor Micali is the recipient of the prestigious Turing Award for computer science, and many of his inventions have directly impacted the cryptocurrency scene (Michael, 2020).

Pago transaction gateway provides a solution for digital payments with the Algorand blockchain. The transaction gateway enables borderless, secure, and fast digital payments for anyone with a smartphone (Gieseke, 2020).

Table 1 and Table 2 detail the software languages, tools, libraries, and versions used and a description that summarizes the logic behind the choice of the technology listed.

Table 1. Software languages

Languages	**Version**	**Description of Choice**
Java	JDK 12	The platform will use Java version JDK 12 will to implement microservices. The platform will use JDK 12 version as Spring cloud libraries are stable and compatible with JDK 12.

Software language choice and a brief description of the option.

Table 2. Software Tools

Tools	**Version**	**Description of Choice**
Spring Boot		Spring Boot provides a set of libraries that facilitate and enhance the features of microservices. Thus, making it the current tool to go when building microservices.
Spring Cloud		Spring Cloud provides a collection of tools that facilitate and provide solutions to common problems when building distributed systems.
Maven	3.6	Maven is a project management tool based on the project object model. It is efficient to build projects, manage dependencies, and simplifies the build process.
Jmeter	2.0	Apache JMeter provides the capabilities for load testing, analyzing, and measuring the performance of services.

Algorand Blockchain	2.0	Algorand is a blockchain project whose goal is to create a transparent system in which everyone can achieve success through decentralized projects and applications. Algorand's consensus mechanism promotes the performance, security, and openness of a decentralized network.
Cassandra	3.11.8	Cassandra database was chosen as the technology to be used to manage the ride-sharing platform data. Cassandra helps solve complicated tasks with ease. It provides rapid writing and even faster reading, extreme resilience, and fault tolerance.
Cadence	0.13.0	Cadence is a distributed, scalable, durable, and highly available orchestration engine to execute asynchronous long-running business logic in a scalable and resilient way.
Pago Transaction Gateway	0.5.1	Pago transaction gateway enables borderless, secure, and fast digital payments using the Algorand blockchain.

Software tools requirements and a brief description of choice.

Chapter VI.

Rideshare Platform Design

This chapter documents the design for the ride-sharing platform. The design sections dive into the technical design details to note how the main components will interact across a proposed microservice-based solution.

The chapter will cover the domain model that classifies the core objects of the design into microservices.

The chapter then covers the modular design sections for each platform's microservices and the modular design section for the android rides-sharing application. The technical details will cover the classes' documentation, design patterns, sequence diagrams, state machine diagrams, and wireframes where applicable for a specific section.

6.1. Domain Model

This section covers the domain model for the ride-sharing platform. The domain model is the base or core of the design to guide the ride-sharing platform's implementation. The domain model shown in Figure 5 covers the main objects identified as a requirement to cover all the use cases and build the ride-sharing platform.

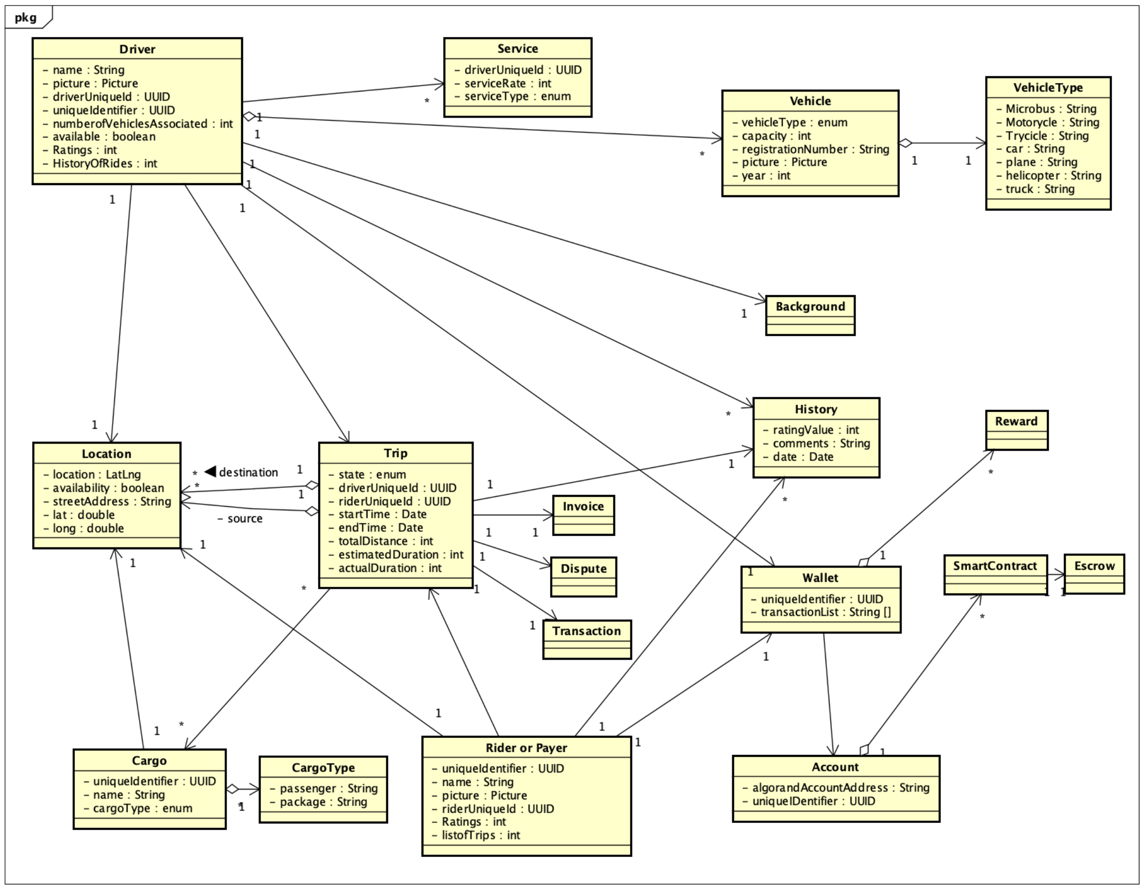


Figure 5. Domain Model

The domain model for the ride-sharing platform.

A package class diagram with a package model classifies the domain model objects into their main categories. Figure 6 presents the object package classification that served as the base for the microservice architecture. The microservice architecture includes the account management microservice, trip management microservice, and wallet management microservice.

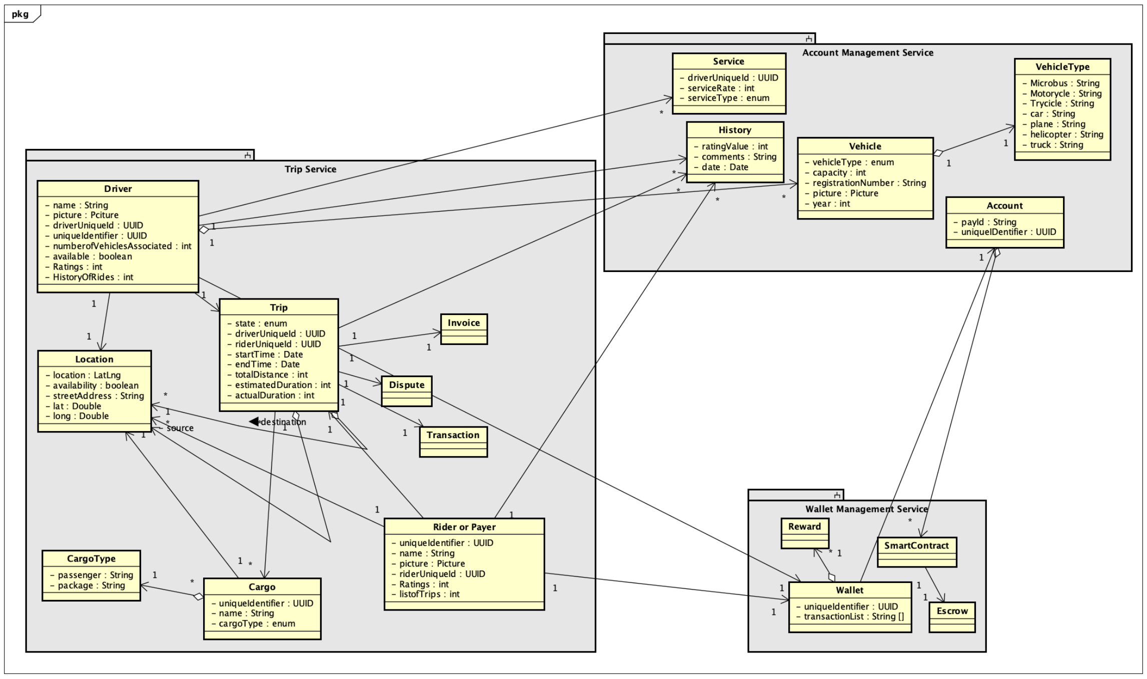


Figure 6. Package Model

Domain Model package classification.

The final package class diagrams show further details on implementing each of the packages outlined in Figure 6. As previously mentioned, each of those packages served as the core of the microservices' design for the ride-sharing platform.

6.2. Account Management Service Design

This section outlines the modular design for the account management service, a microservice component of the ride-sharing platform. The account management service is the microservice that fulfills the functionalities of account creation and account management. The service implements the APIs to service the account creation and account management requests of the android application. The account management requests handled from the account management microservice must cover account creation requirements, account authentication, and link a Pago account for both riders and drivers. The service must write and read from a database to persist the user's data to service the account creation and account authentication requests.

6.2.1. Class Diagram

Figure 7 illustrates the detailed class design to stand up a spring boot microservice with the Casandra database. The diagram includes the required classes and associations to meet the requirements to create and manage accounts for the ride-sharing platform.

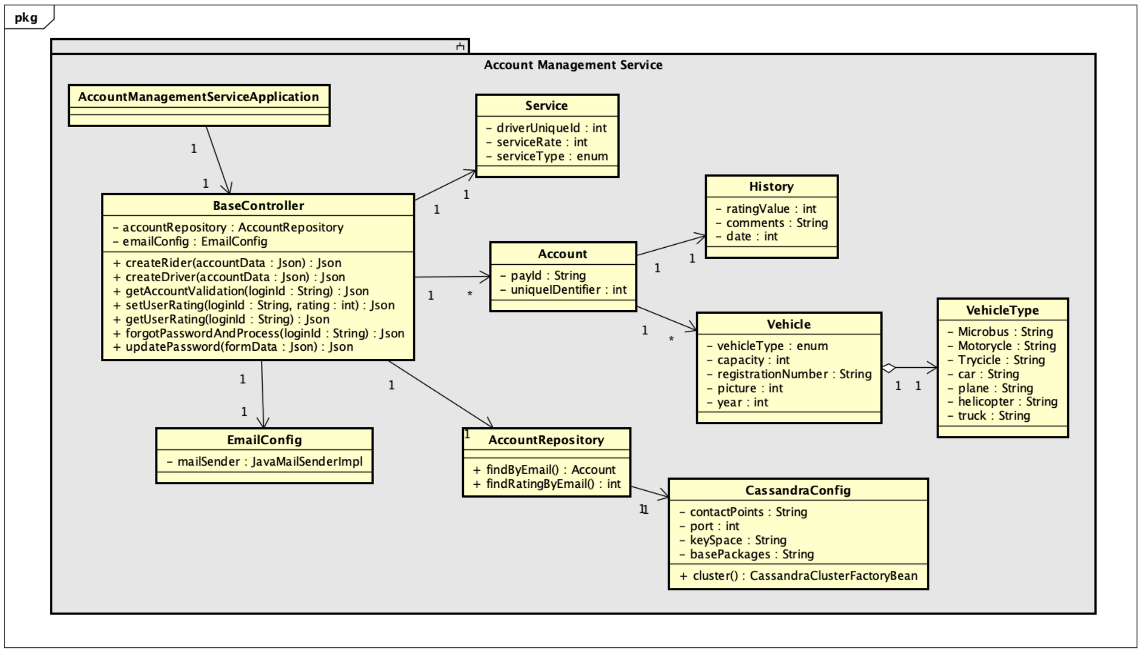

pkg
Account Management Service
AccountManagementServiceApplication
Service
- driverUniqueId : int
- serviceRate : int
- serviceType : enum
History
- ratingValue : int
- comments : String
- date : int
BaseController
- accountRepository : AccountRepository
- emailConfig : EmailConfig
+ createRider(accountData : Json) : Json
+ createDriver(accountData : Json) : Json
+ getAccountValidation(loginId : String) : Json
+ setUserRating(loginId : String, rating : int) : Json
+ getUserRating(loginId : String) : Json
+ forgotPasswordAndProcess(loginId : String) : Json
+ updatePassword(formData : Json) : Json
Account
- payId : String
- uniqueIDentifier : int
Vehicle
- vehicleType : enum
- capacity : int
- registrationNumber : String
- picture : int
- year : int
VehicleType
- Microbus : String
- Motorycle : String
- Trycicle : String
- car : String
- plane : String
- helicopter : String
- truck : String
EmailConfig
- mailSender : JavaMailSenderImpl
AccountRepository
+ findByEmail() : Account
+ findRatingByEmail() : int
CassandraConfig
- contactPoints : String
- port : int
- keySpace : String
- basePackages : String
+ cluster() : CassandraClusterFactoryBean
1
*

Figure 7. Account Management Microservice Class diagram

The class diagram illustrates the main classes for the account management microservice.

The AccountManagementService class is the main class that the ride-sharing platform calls upon staring at the account management microservice application. It contains the SpringBootApplication annotation, which identifies the class as the single entry point to bootstrap and launches the spring boot application.

The CassandraConfig class extends AbstractCassandraConfiguration and is in charge of specifying the configuration for the Cassandra database. Cassandra's structure contains the contact points, Cassandra port, and keyspace name for the account management service database. The AccountRepository class uses those configurations to create an account repository model as a table for the Cassandra database.

The EmailConfig class is in charge of the implementation to configure and send an email to cover the scenario where a user has forgotten the password to access the ride-sharing platform. The user must reset the password to be able to regain access.

The BaseController class is the class that is in charge of implementing the methods that provide the REST interfaces that the ride-sharing Android application calls to execute all the actions required to service the account creation and account management. The class diagram, as illustrated in Figure 8, shows the methods that service those requests.

6.2.2. Design Details

The account management microservice follows the MVC design pattern. The service uses Spring Boot framework, which structures components into three main buckets to pursue the separation of concerns concept, which is the core principle of the MVC design pattern. The model is the first bucket, and its purpose is to model the application objects, store, and manage data. The view bucket covers a graphical user interface representation. The controller bucket, which is the core class of the application, connects the model and view buckets. The controller class receives input from the view and uses logic to translate the information into a model demand. The model grabs the data, and the controller passes the data from the model back to the user's view to see it displayed (Doherty, 2020).

The account management microservice manages the data required to fulfill the account creation requests and account management requests. The account management data required to manage the rider and driver accounts consists of a login name, encrypted password, and linked Pago account identification. The microservice stores account management data in its dedicated Cassandra database.

6.2.3. Sequence Diagram

The sequence diagram in Figure 8 illustrates how the request to create an account is initiated from the ride-sharing android application and processed across the account management microservice. The account management service then stores the account information into the rideshare account table on the Cassandra database.

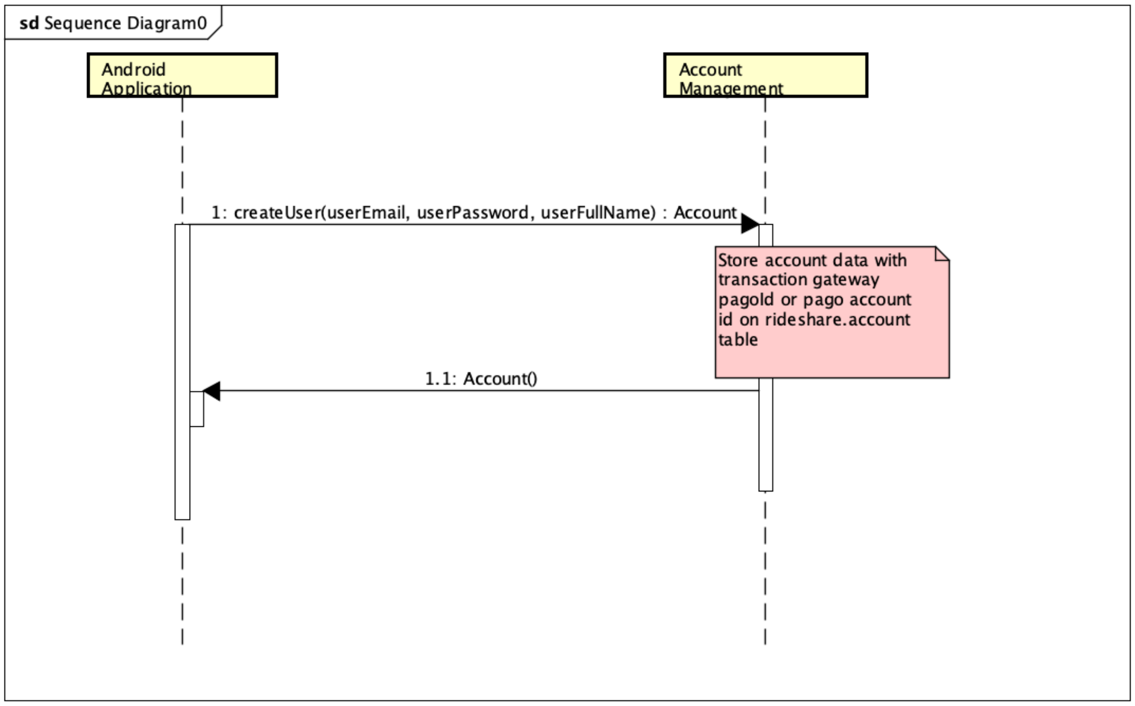

sd Sequence Diagram0
Android Application
Account Management
1: createUser(userEmail, userPassword, userFullName) : Account
Store account data with transaction gateway pagoId or pago account id on rideshare.account table
1.1: Account()

Figure 8. Create an account sequence diagram

The create account sequence diagram shows the message that covers the happy path flow to create a rider account with the platform. Both driver and rider follow the same sequence diagram. The only difference is the type of account and additional attributes for the driver that the ride-sharing application is sending, such as license, registration, and vehicle type.

6.3. Trip Management Service Design

This section describes the design of the trip management service, a microservice component for the ride-sharing platform. A trip management microservice implementation is required to manage the trip for both the drivers and the users seeking the ride or package delivery service. The trip management microservice must implement the APIs to service the trip's orchestration due to the Android application's requests. The trip management microservice must have the functionality to write and read to a database.

A database must implement the tables to manage the data for the trip location and destination.

6.3.1. State Machine Diagrams

This section introduces the design of the ride-sharing platform state machine diagrams. The state machine diagrams serve as the primary guide to implement the orchestration for the trip management service. Figure 9 shows the driver state machine.

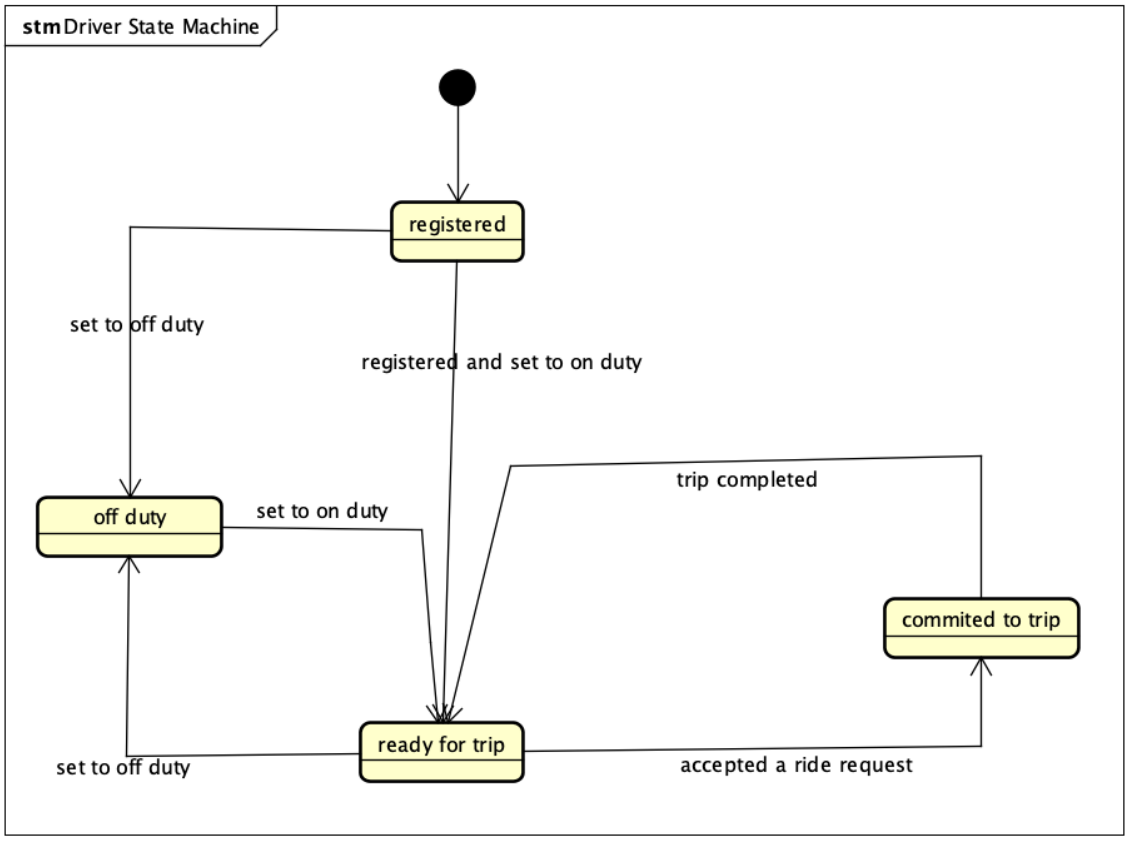


Figure 9. Driver State Machine

Driver state machine.

The trip state machine covers all the states required to complete a trip execution. The orchestrator manages the trip execution by executing all the steps on the trip state machine. Figure 10 presents the states that the orchestrator must evaluate to meet trip completion requirements.

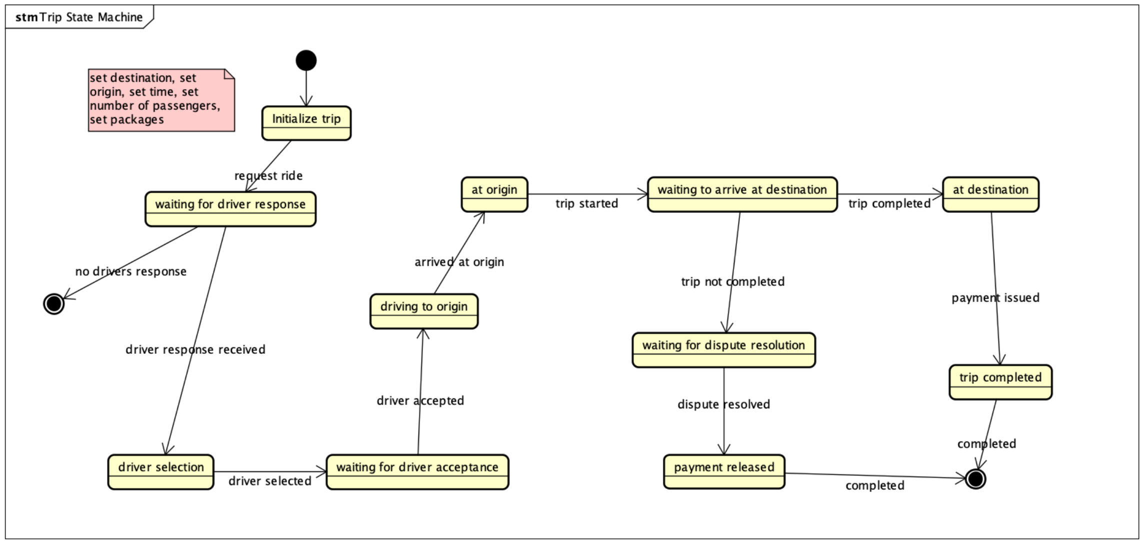


Figure 10. Trip State Machine

Required states that the orchestrator must evaluate for trip completion.

6.3.2. Class Diagram

The class diagram in Figure 11 illustrates the detailed class design implemented to stand up a spring boot microservice with the Cassandra database to meet the requirements to manage the ride-sharing platform's trip. The trip management service is the module in charge of the orchestration of the ride-sharing platform's trip. The OrchestrationImpl class handlers the orchestration of the trip is handled by the OrchestrationImpl class, whose purpose is to encapsulate all the data required for performing the actions or commands needed to complete a given trip.

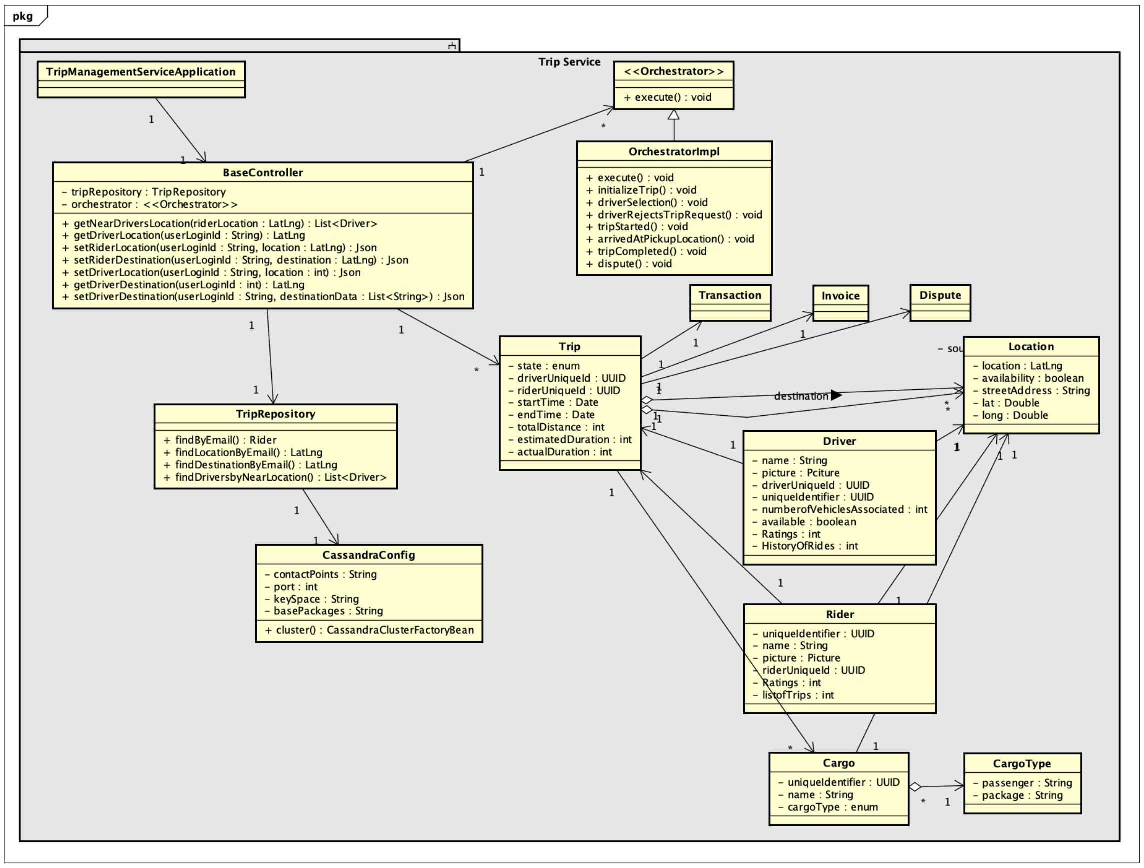

pkg
Trip Service
TripManagementServiceApplication
BaseController
- tripRepository : TripRepository
- orchestrator : <<Orchestrator>>
+ getNearDriversLocation(riderLocation : LatLng) : List<Driver>
+ getDriverLocation(userLoginId : String) : LatLng
+ setRiderLocation(userLoginId : String, location : LatLng) : Json
+ setRiderDestination(userLoginId : String, destination : LatLng) : Json
+ setDriverLocation(userLoginId : String, location : int) : Json
+ getDriverDestination(userLoginId : int) : LatLng
+ setDriverDestination(userLoginId : String, destinationData : List<String>) : Json
<<Orchestrator>>
+ execute() : void
OrchestratorImpl
+ execute() : void
+ initializeTrip() : void
+ driverSelection() : void
+ driverRejectsTripRequest() : void
+ tripStarted() : void
+ arrivedAtPickupLocation() : void
+ tripCompleted() : void
+ dispute() : void
Transaction
Invoice
Dispute
Trip
- state : enum
- driverUniqueId : UUID
- riderUniqueId : UUID
- startTime : Date
- endTime : Date
- totalDistance : int
- estimatedDuration : int
- actualDuration : int
Location
- location : LatLng
- availability : boolean
- streetAddress : String
- lat : Double
- long : Double
- sou
destination
TripRepository
+ findByEmail() : Rider
+ findLocationByEmail() : LatLng
+ findDestinationByEmail() : LatLng
+ findDriversbyNearLocation() : List<Driver>
CassandraConfig
- contactPoints : String
- port : int
- keySpace : String
- basePackages : String
+ cluster() : CassandraClusterFactoryBean
Driver
- name : String
- picture : Pciture
- driverUniqueId : UUID
- uniqueIdentifier : UUID
- numberofVehiclesAssociated : int
- available : boolean
- Ratings : int
- HistoryOfRides : int
Rider
- uniqueIdentifier : UUID
- name : String
- picture : Picture
- riderUniqueId : UUID
- Ratings : int
- listofTrips : int
Cargo
- uniqueIdentifier : UUID
- name : String
- cargoType : enum
CargoType
- passenger : String
- package : String
1
*

Figure 11. Trip Management Microservice Class diagram

A class diagram that illustrates the main classes for the trip management microservice.

The CassandraConfig class and TripRepository class follow the same structure as the account and wallet management microservice. The difference is that the object modeled by the TripRepository class and Cassandra table is that of a trip object. It is worth mentioning that the TripRepository stores all the information required to orchestrate a given trip, such as location and destination.

Similar to the implementation of the other two microservices, the BaseController class is the class that is in charge of implementing the methods that provide the rest interfaces that the ride-sharing android application calls to service requests. The BaseController delivers the APIs for a user or rider to get the near drivers, set a driver location, get a driver location, set a rider location, and get a rider location. The android ride-sharing application's request to those APIs will then initiate a given controller API with the orchestrator to handle all the trip logic.

6.3.3. Design Details

The trip management service is the primary contact for the android ride-sharing application to service trip orchestration. The trip management microservice is a spring boot java based application that follows the MVC pattern. An orchestrator module knows the steps needed to complete a trip and is in charge of handling trip orchestration logic. If anything fails along with a particular trip execution, the orchestrator is responsible for coordinating the rollback by sending the commands to undo the previous operation. It does so by following a command design pattern.

In the command pattern, the application declares an abstract command class as an interface for executing operations. This command class defines a method for execution, which the service needs to implement for each concrete command (Caballero, 2019).

The orchestrator module declares the abstract command class as an interface for executing operations for the current implementation scope. The interface defines an execute method, and the orchestration implementation, or OrchestrationImpl class, defines the trip execution commands. The execute method serves as a bridge between a receiver object and an action. The OrchestrationImpl class ensures that the ride-sharing platform completes the necessary states for a given trip completion by implementing this approach.

The trip management data required to orchestrate a specific trip consists of location, destination, and package delivery data. The trip management service stores the trip management data in the microservice dedicated Cassandra database.

6.3.4. Sequence Diagram

Figure 12 presents the initiation of a given trip from a rider request. A given trip’s initiation entitles the android ride-sharing application to find the driver as available for trip requests. A rider has initiated a request for a trip by selecting a destination. The rider has found the driver. The sequence diagram below shows how the rider starts the trip request by setting the destination from the rider's android application. The getDriverDestination method is polling and called upon the driver’s android application until a driver finds a destination. If the driver finds a destination, the orchestrator can proceed to initiate the trip.

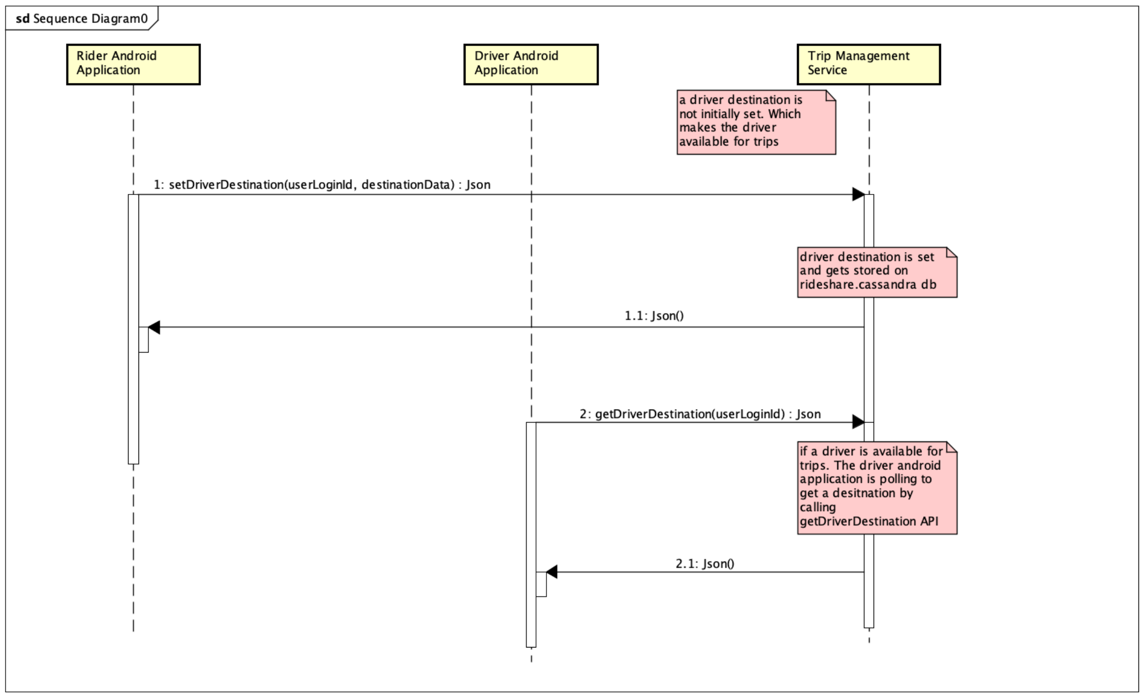

sd Sequence Diagram0
Rider Android Application
Driver Android Application
Trip Management Service
a driver destination is not initially set. Which makes the driver available for trips
1: setDriverDestination(userLoginId, destinationData) : Json
driver destination is set and gets stored on rideshare.cassandra db
1.1: Json()
2: getDriverDestination(userLoginId) : Json
if a driver is available for trips. The driver android application is polling to get a desitnation by calling getDriverDestination API
2.1: Json()

Figure 12. Rider issues a trip request

The rider issues a trip request.

6.4. Wallet Management Service Design

This section outlines the design of the wallet management service, a microservice component of the ride-sharing platform. The wallet management microservice must implement the APIs to service the wallet balance and payment transactions via the Pago transaction gateway. Actions from the android application will initiate the wallet API requests. The wallet management microservice must write and read to a database. The database must implement the table to write and read the Pago account identification for the user. The ride-sharing platform will use the Pago account identification as the single identification factor to get the wallet balance and issue unsigned payment transactions via the transaction gateway.

6.4.1. Class Diagram

The class diagram in Figure 13 illustrates the detailed class design implemented to stand up a spring boot microservice with Casandra database to meet the requirements to manage the ride-sharing platform’s wallet. The wallet is mostly a module that serves as a pass-through to interact with the transaction gateway to help the ride-sharing platform’s transaction requests and provide a point of contact with Pago's transaction gateway.

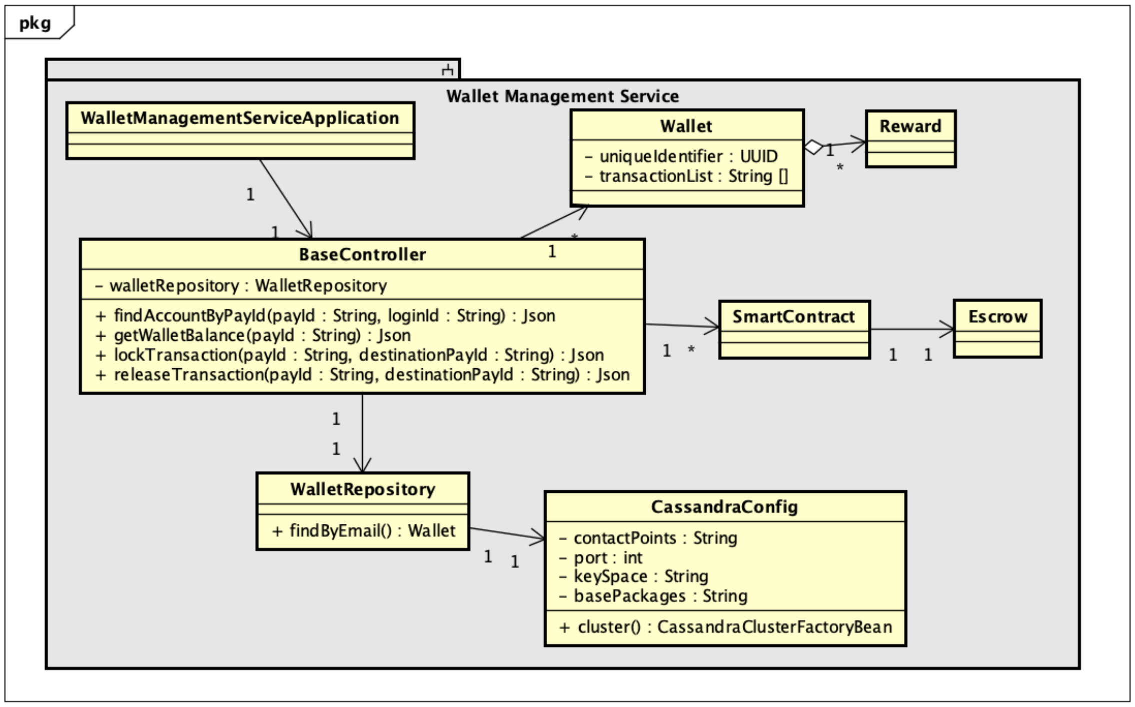

pkg
Wallet Management Service
WalletManagementServiceApplication
Wallet
- uniqueIdentifier : UUID
- transactionList : String []
Reward
BaseController
- walletRepository : WalletRepository
+ findAccountByPayId(payId : String, loginId : String) : Json
+ getWalletBalance(payId : String) : Json
+ lockTransaction(payId : String, destinationPayId : String) : Json
+ releaseTransaction(payId : String, destinationPayId : String) : Json
SmartContract
Escrow
WalletRepository
+ findByEmail() : Wallet
CassandraConfig
- contactPoints : String
- port : int
- keySpace : String
- basePackages : String
+ cluster() : CassandraClusterFactoryBean

Figure 13. Wallet Management Microservice Class diagram

The class diagram illustrates the main classes for wallet management microservice.

There are two main functions that the wallet microservice must provide to the ride-sharing platform: get the wallet balance for an Algorand address via the Pago identification and sign a transaction via Algorand address via the Pago or payId. The main functions reside on the BaseController class via requests that service the android ride-sharing application. The BaseController initiates calls to the transaction gateway APIs defined on the Pago Transaction Gateway API specification (Gieseke, 2020).

The CassandraConfig class and WalletRepository class follow the same structure as the account management microservice. The difference is that the object modeled by the WalletRepository class Cassandra table is that of a wallet object.

The wallet management microservice uses a smart contract object to model the contracts required to process the ride-sharing platform's transactions. The smart contract locks a transaction by getting the funds for a particular trip from the user or rider once a trip pick-up location and destination is selected. The smart contract will then release those funds to the driver if the driver completes both the pick-up and trip destination execution. The smart contract instantiates an escrow object to record the events for a given trip.

6.4.2. Design Details

The wallet management microservice is a microservice component of the platform. The wallet management microservice is a spring boot java-based application that follows the MVC pattern. The wallet management microservice is in charge of interacting with the Pago transaction gateway to service any transaction requests from the android ride-sharing application. Two main interactions happen as part of servicing a transaction from clients: getting a client wallet balance and issuing unsigned wallet transactions. The wallet management data for a given user account consists of the Pago account identification. The wallet management service stores the Pago account identification in its dedicated Cassandra database.

6.4.3. Sequence Diagram

The sequence diagram in Figure 14 shows the request performed to issue payment after trip completion. The wallet management service issues payment by sending a request to Pago's transaction gateway. The smart contract module on the wallet management application must acknowledge or have flags as true for the driver arriving at the pick-up location and the driver arriving at the destination to identify the trip as completed. The implementation details and future work section discuss more information about the smart contract implementation.

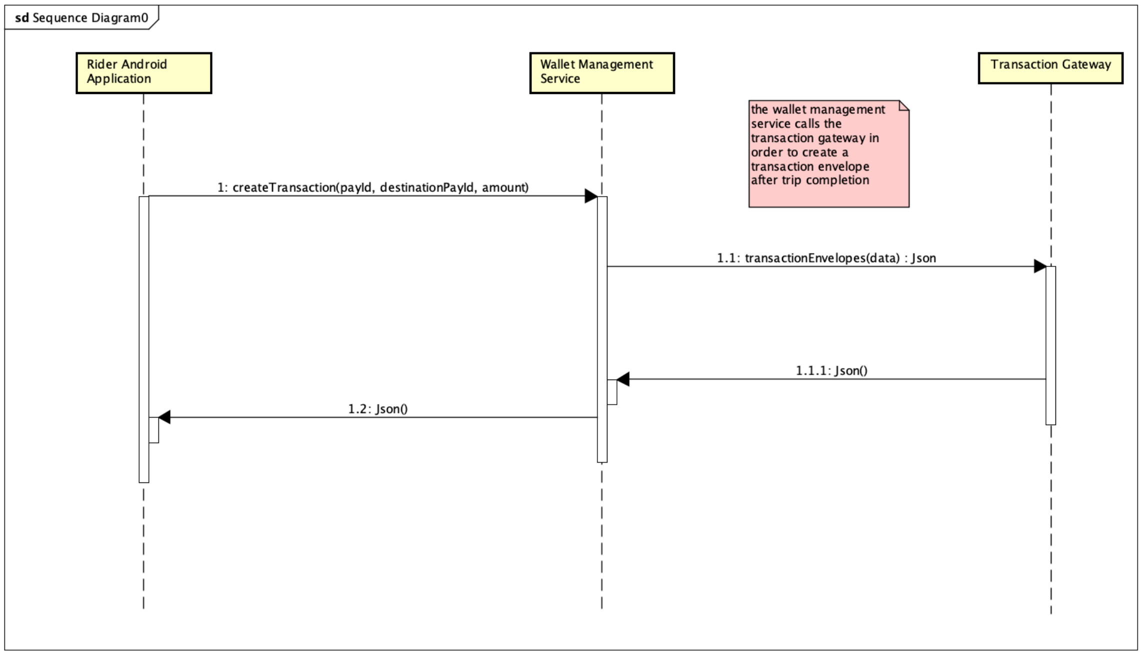

sd Sequence Diagram0
Rider Android Application
Wallet Management Service
Transaction Gateway
the wallet management service calls the transaction gateway in order to create a transaction envelope after trip completion
1: createTransaction(payId, destinationPayId, amount)
1.1: transactionEnvelopes(data) : Json
1.1.1: Json()
1.2: Json()

Figure 14. Payment issued to the driver

The ride-sharing platform issues a payment from the rider's Pago identification to the driver's Pago identification with the amount specified as the charge for trip completion.

The diagram in Figure 15 illustrates the external requests required to complete a transaction via Pago's transaction gateway due to a payment issued to the driver.

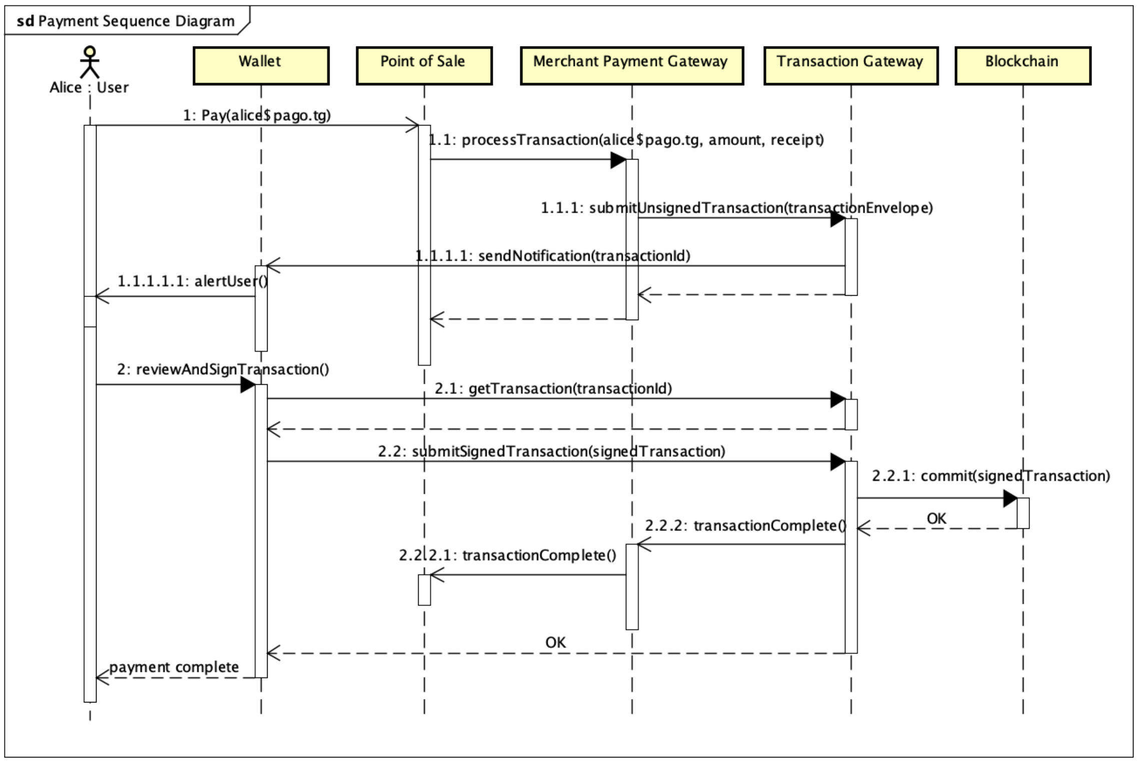

sd Payment Sequence Diagram
Alice : User
Wallet
Point of Sale
Merchant Payment Gateway
Transaction Gateway
Blockchain
1: Pay(alice$pago.tg)
1.1: processTransaction(alice$pago.tg, amount, receipt)
1.1.1: submitUnsignedTransaction(transactionEnvelope)
1.1.1.1: sendNotification(transactionId)
1.1.1.1.1: alertUser()
2: reviewAndSignTransaction()
2.1: getTransaction(transactionId)
2.2: submitSignedTransaction(signedTransaction)
2.2.1: commit(signedTransaction)
OK
2.2.2: transactionComplete()
2.2.2.1: transactionComplete()
OK
payment complete

Figure 15. Transaction Gateway sequence diagram

Transaction gateway sequence diagram that contains the interaction between the gateway and blockchain.

For each trip completed, the ride-sharing platform creates a transaction and submits it to the Transaction Gateway, which manages the signing process and notifies the point-of-sale application when the blockchain successfully commits the transaction. The Transaction Gateway coordinates the interaction between the business application and the Algorand blockchain. The point-of-sale app sends unsigned transactions to the Transaction Gateway, which then forwards the transactions to the appropriate wallets for signing. After signing the transaction, the wallet sends the signed transaction back to the Transaction Gateway, which commits it to the Algorand blockchain. The Transaction

Gateway notifies the point-of-sale application when the payment is complete (Gieseke, 2020).

6.4.4. Payment Workflow

Furthermore, Figure 16 illustrates Pago's payment workflow that takes place to complete a signed transaction.

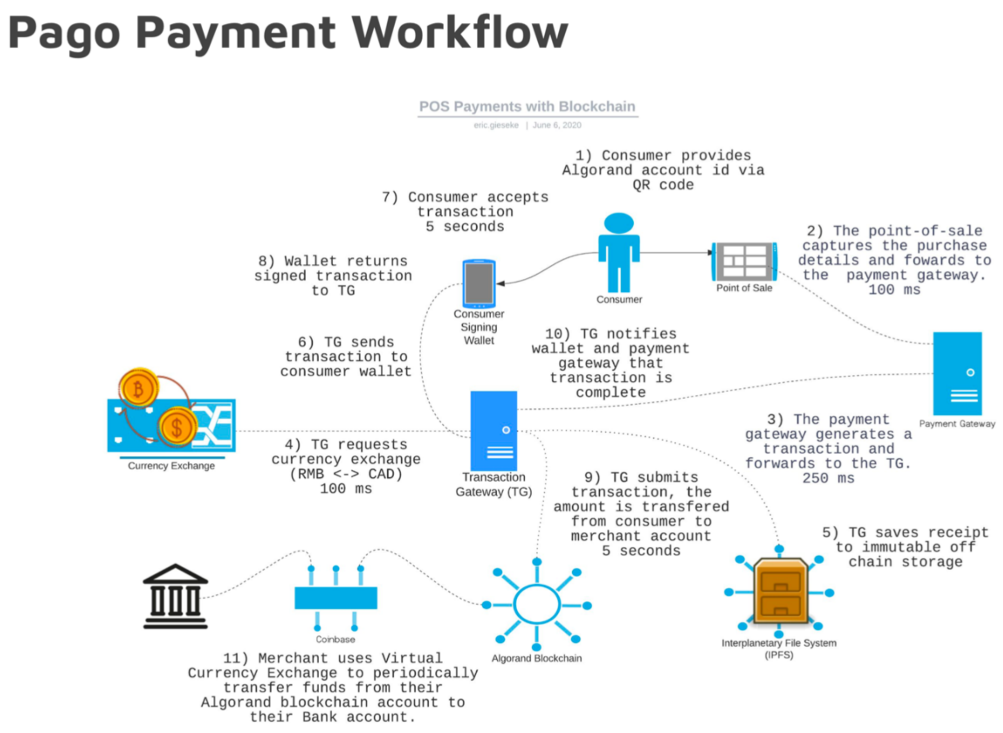

Pago Payment Workflow
POS Payments with Blockchain
eric.gieseke | June 6, 2020
1) Consumer provides Algorand account id via QR code
7) Consumer accepts transaction 5 seconds
2) The point-of-sale captures the purchase details and fowards to the payment gateway. 100 ms
8) Wallet returns signed transaction to TG
Consumer Signing Wallet
Consumer
Point of Sale
6) TG sends transaction to consumer wallet
10) TG notifies wallet and payment gateway that transaction is complete
Payment Gateway
3) The payment gateway generates a transaction and forwards to the TG. 250 ms
Currency Exchange
4) TG requests currency exchange (RMB <-> CAD) 100 ms
Transaction Gateway (TG)
9) TG submits transaction, the amount is transfered from consumer to merchant account 5 seconds
5) TG saves receipt to immutable off chain storage
Coinbase
Algorand Blockchain
Interplanetary File System (IPFS)
11) Merchant uses Virtual Currency Exchange to periodically transfer funds from their Algorand blockchain account to their Bank account.

Figure 16. Pago payment workflow (Gieseke, 2020)

Pago's payment workflow

6.5. Android Ride-sharing Application Design

This section describes the Android ride-sharing application's design, a java-based android application for the ride-sharing platform. The ride-sharing application must interact with the platform microservices by making requests via rest API calls to achieve those functionalities. The ride-sharing application is a java based android application whose purpose is to serve as the interface for the users to interact with the ride-sharing platform. The ride-sharing android application makes rest API calls to request actions to the microservices that make up the platform as a whole.

The ride-sharing platform does not show the view as part of the microservices codebase or classes. The android ride-sharing android application shows the view as a function of the ride-sharing android application due to receiving responses from the microservices.

6.5.1. Screen Captures

The wireframe section illustrates the ride-sharing android application user interfaces implemented to serve as the point of contact to initiate actions by the users, riders, or drivers to interact with the ride-sharing platform.

Figure 17 illustrates the android interface that the android ride-sharing application presents to the user upon launch. The android ride-sharing application shows the option to ride or drive to the user.

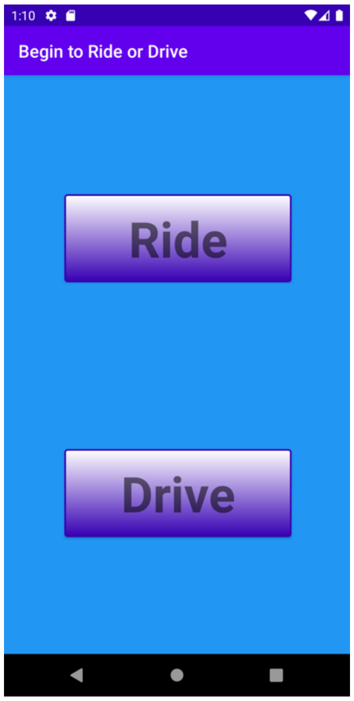


Figure 17. Launch screen

Launch screen for the ride-sharing application.

On the left, Figure 18 illustrates the android interface that is shown upon a user selecting the option to ride. The android ride-sharing application presents the user the option to create an account or signing into the ride-sharing application. Similarly, on the right, Figure 19 illustrates the options provided if a user selects to drive.

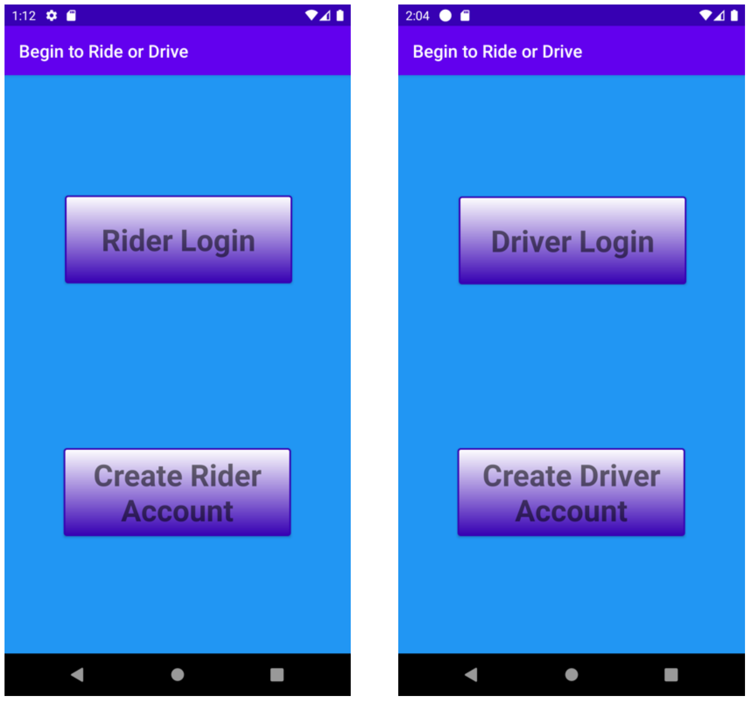


Figure 18. Main screen

The rider or driver has the option to create an account or log in to the ride-sharing application.

Figure 19 below illustrates the android interface that is shown upon a user selecting to login to the ride-sharing account. The user is presented with the form to log in to the platform. If authentication with the platform fails, the android ride-sharing application will show the user the option to register with the platform.

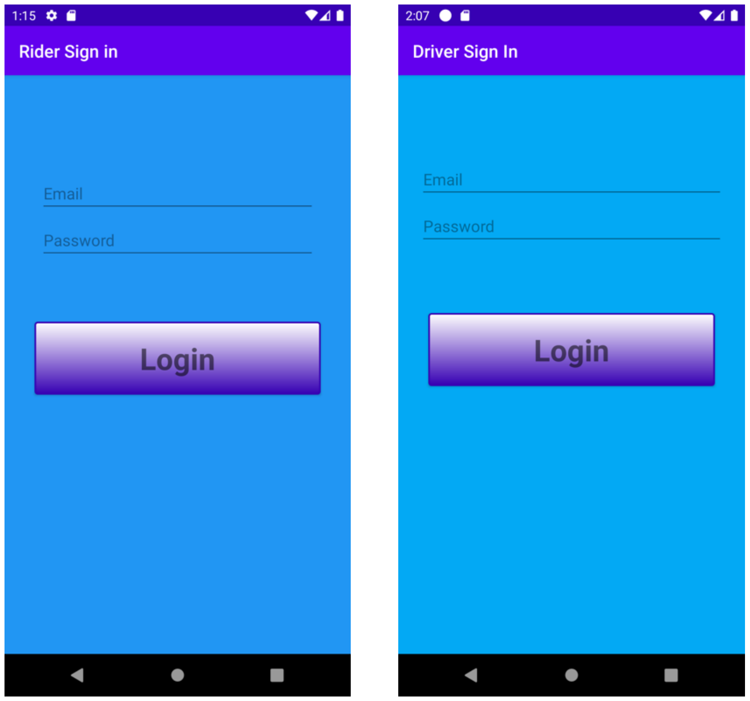


Figure 19. Rider and driver login screen

The rider and driver login interfaces.

Figure 20 illustrates the android interface that is shown upon a user selecting to create an account. The android ride-sharing application presents the user the option to link a Pago account by providing the Pago account identification. The application then requires additional information to create a driver account. The application shows prospective drivers with a second form requiring a license, registration identification number, and vehicle type for the driver account creation.

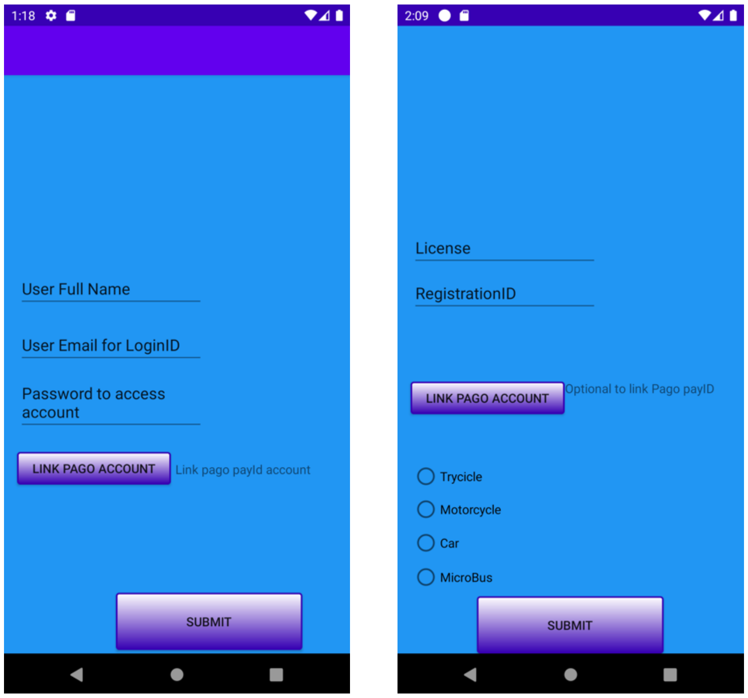


Figure 20. Create an account

Option to create an account.

Figure 21 illustrates the android interface shown upon a user is authenticated or has created an account on the ride-sharing platform. The android ride-sharing application presents the user with an interface that provides a map. The interactive map shows the user location, a wallet button, and the user's fragment to select a destination if the user is a rider. If the user is a driver, the android ride-sharing application presents the user with the option to search for riders.

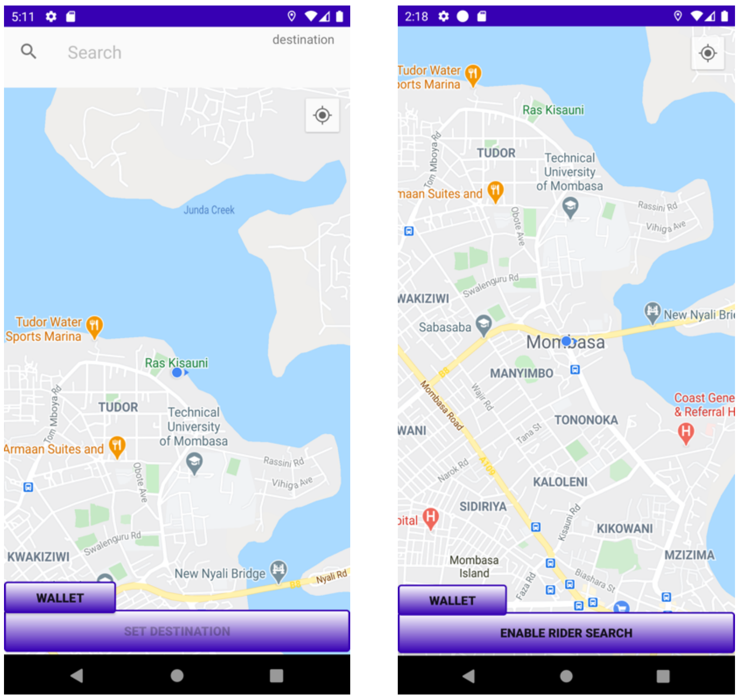


Figure 21. Welcome map

Rider and driver map after authentication or account creation.

The wallet provides the options to get the Pago account balance and to show the Pago account identification for the user, as shown in Figure 22. The android ride-sharing application disables the set destination button by default and will be enabled once the user selects a destination.

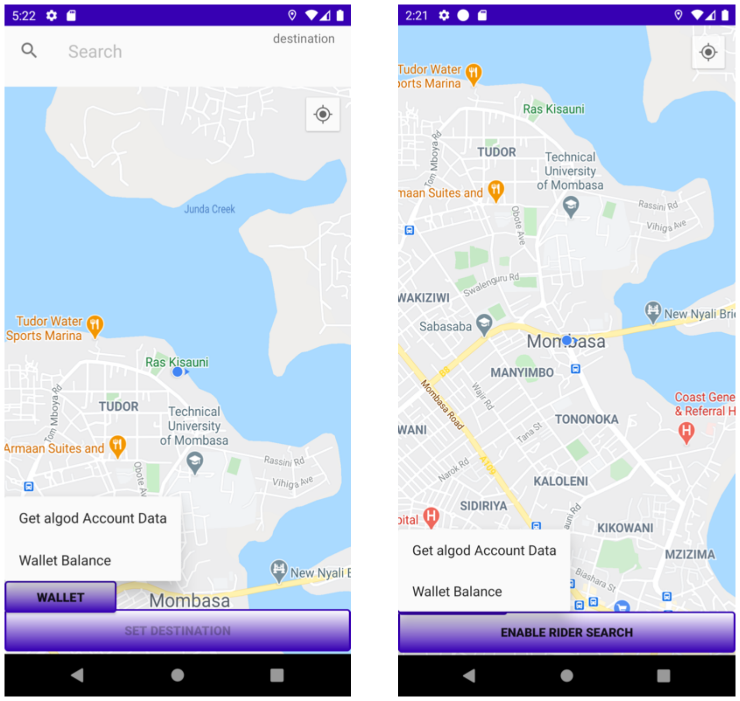


Figure 22. Wallet options

The interactive map with the wallet options menu.

The android ride-sharing application will present the rider with the option to modify the pick-up location. If that option is selected, the android ride-sharing application will allow the rider to choose package delivery. Once those options are selected, the rider can choose to start searching for available drivers. Figure 23 presents the rider map options.

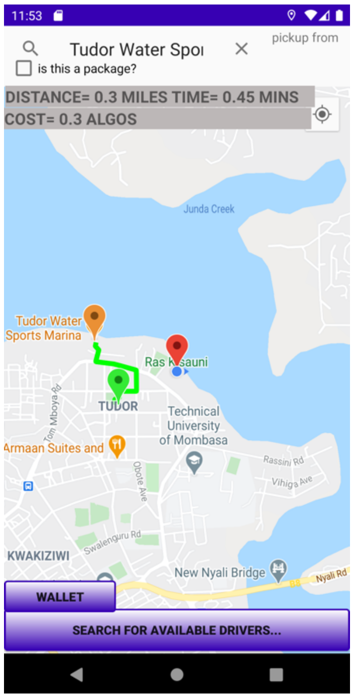
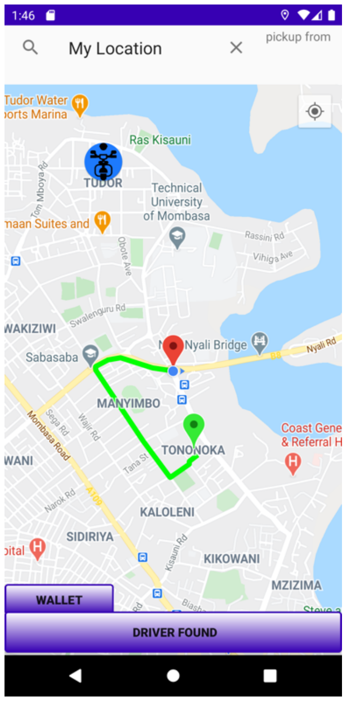


Figure 23. Rider map options

Rider map showing pick-up location and package option available for the user.

Suppose the trip management service has found a driver. In that case, the android ride-sharing application presents an icon showing the driver's location and specifying the driver-vehicle type to the rider that made the request. If the driver icon is selected, the android ride-sharing application presents the driver's ratings for the user who requested to proceed with the trip request. If the user chooses not to proceed with the trip request with the given driver, they can search for drivers again. The android ride-sharing application can find multiple drivers on the map for a ride's request. The trip management service has a configuration to specify the maximum number for drivers to search for a driver search request. Figure 24 illustrates the details presented to the users.



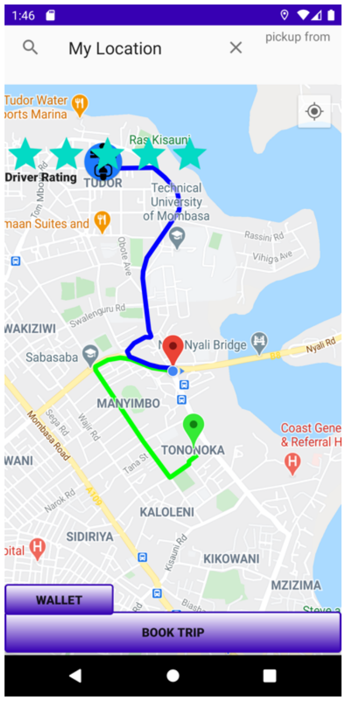


Figure 24. The ride-sharing application finds a driver as a result of the trip request

The ride-sharing application finds a driver upon rider request.

Similarly to the rider's information, the driver is also presented with the rider's rating and given the option to reject a trip request. If the driver accepts the trip request by selecting the Book Trip button, the android ride-sharing application presents the driver with the option to launch directions on google maps.

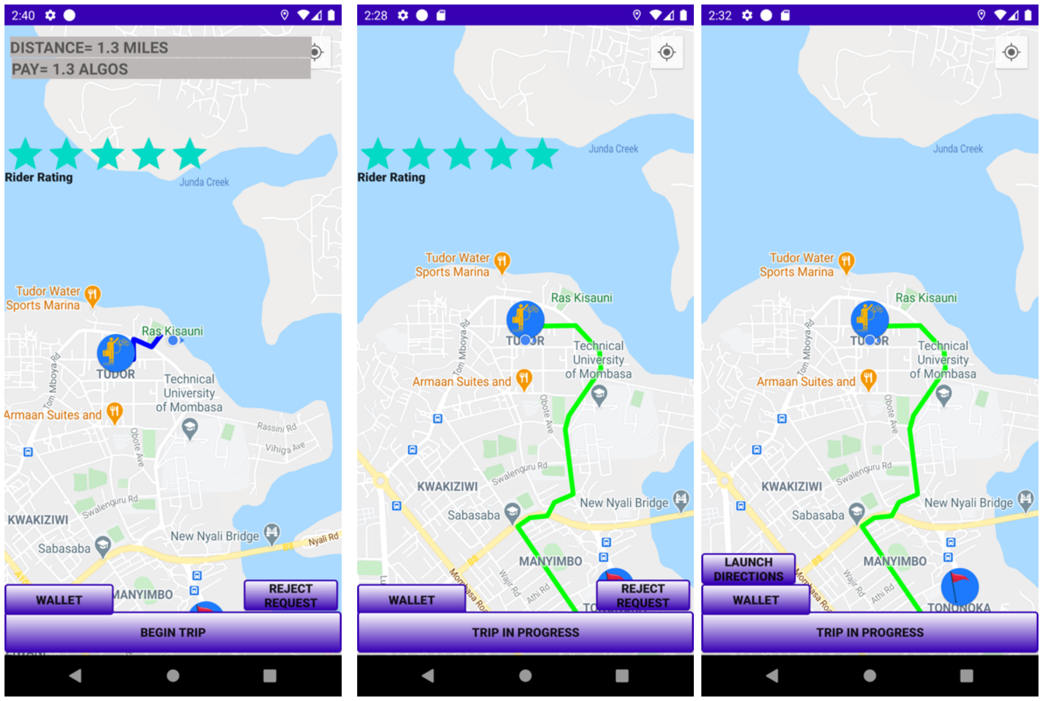
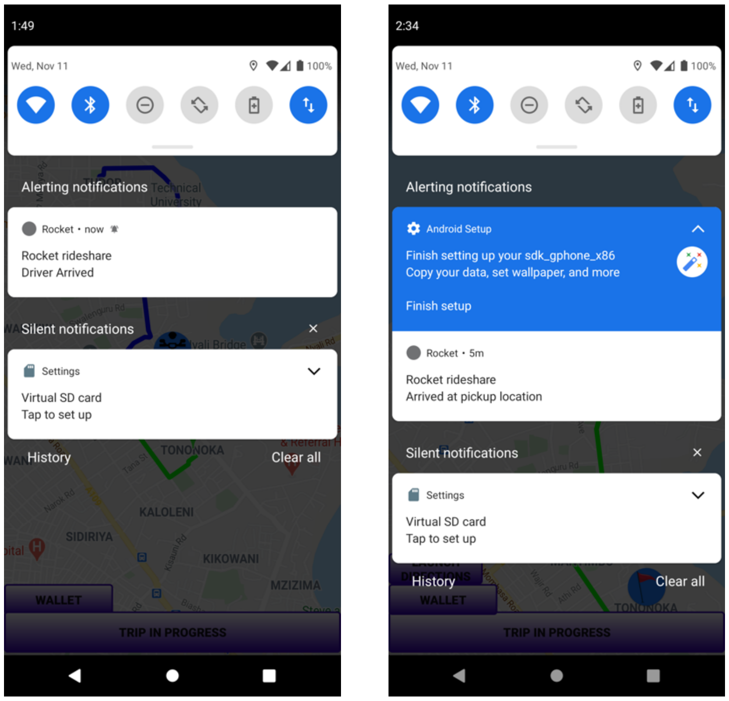


Figure 25. Driver trip options

The android ride-sharing application presents the rider's rating to the driver. Upon acknowledging the rating, the driver has the option to reject the request. Upon accepting a trip request, the driver has the option to launch directions on google maps.

Once a driver arrives at the selected pick-up location, both the rider and driver will receive a notification on their respective phones, as illustrated in Figure 26.


1:49

Wed, Nov 11 100%

Alerting notifications

Rocket • now

Rocket rideshare
Driver Arrived

Silent notifications

Settings

Virtual SD card
Tap to set up

History Clear all

WALLET

TRIP IN PROGRESS

2:34

Wed, Nov 11 100%

Alerting notifications

Android Setup

Finish setting up your sdk_gphone_x86
Copy your data, set wallpaper, and more

Finish setup

Rocket • 5m

Rocket rideshare
Arrived at pickup location

Silent notifications

Settings

Virtual SD card
Tap to set up

History Clear all

WALLET

TRIP IN PROGRESS

Figure 26. A driver arrives at the pick-up location

The ride-sharing application issues a notification to the driver and rider phone upon the driver's arrival at the pick-up location.

Once a driver arrives at the destination, both the rider and driver get a notification. Both the rider and driver are also shown the option to rate each other after trip completion. The payment for the trip is issued right after the trip is completed. The driver has the opportunity to verify payment completion by selecting to see the wallet balance right after trip completion.

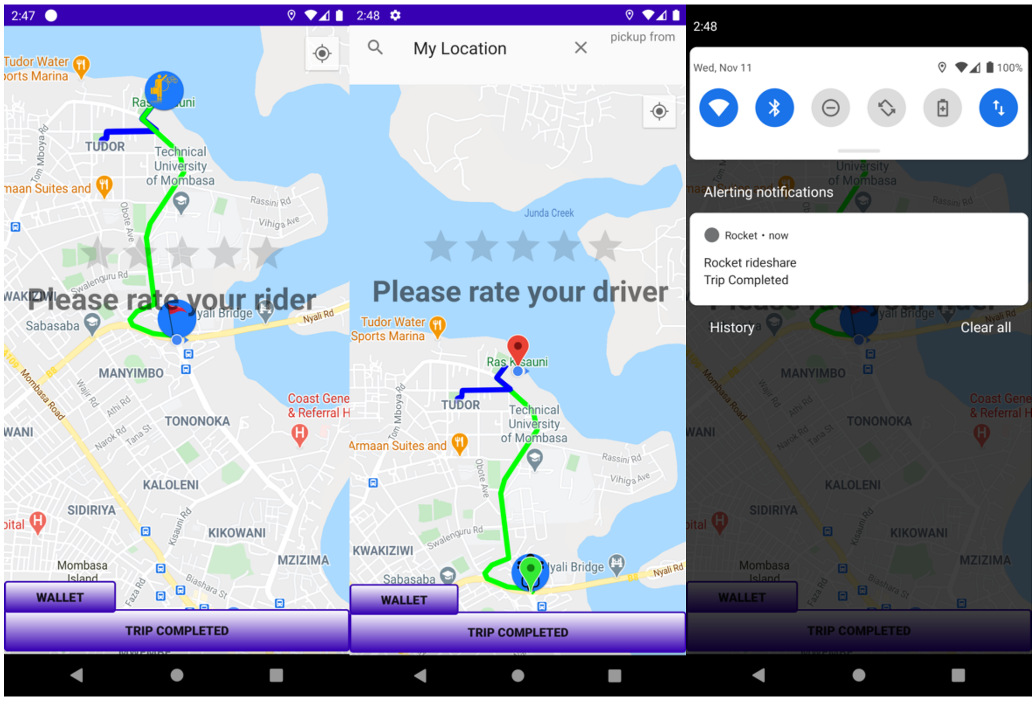


Figure 27. Trip complete

The options to rate rider and driver. The ride-sharing application issues a notification after trip completion.

Chapter VII.

Deployment

With an infrastructure in place, the project then established a deployment process to publish the ride-sharing platform and make a beta version available for users to test. It is worth mentioning that the strategy to deploy the ride-sharing platform revolved around targeting users in Kenya since, on the design and gather requirements phase, the initial research identified the need and interest of a ride-sharing solution from the Drivers Consortium. The vast or most users in Kenya use android phones; therefore, the target was to deploy the ride-sharing platform in Kenya and make it available for android devices.

The microservice components were deployed to Digital Ocean, an infrastructure cloud service provider. Digital Ocean is a unique cloud hosting provider that offers cloud computing services to business entities to scale themselves by deploying Digital Ocean applications that run parallel across multiple cloud servers without compromising performance. The droplets that Digital Ocean offers have a lightning-quick boot time that slates in about 55 seconds. Digital Ocean servers are on powerful Hex Core machines with dedicated ECC Ram and RAID SSD storage. Furthermore, it offers private networking among the VMs for running database clusters and distributed systems in a few selected regions (Saleem, 2019).

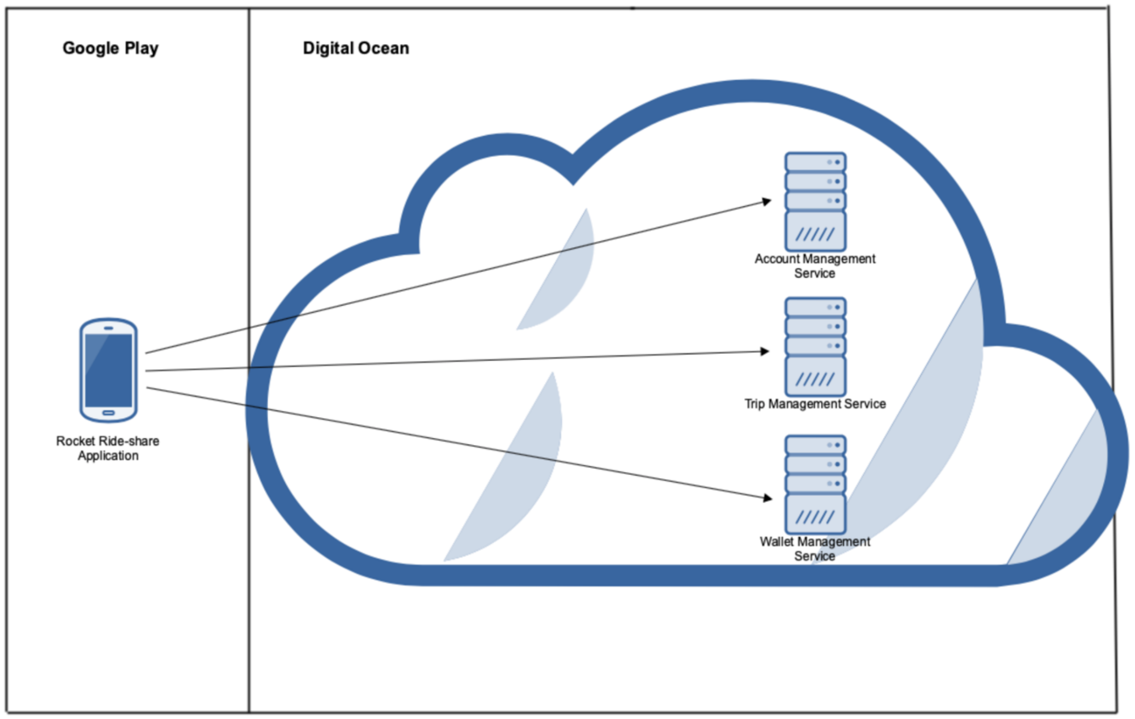

Google Play
Digital Ocean
Rocket Ride-share Application
Account Management Service
Trip Management Service
Wallet Management Service

Figure 28. Ridesharing platform infrastructure

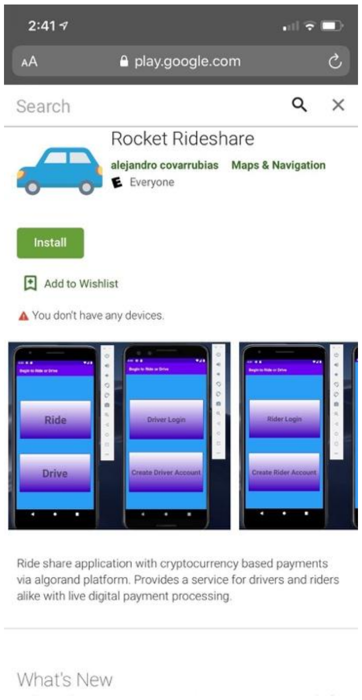
The ride-sharing platform infrastructure encompasses cloud servers running on Digital Ocean. Each of the servers runs its own Cassandra database on Digital Ocean and the ride-sharing application available on the google play store.

As previously mentioned, the ride-sharing application was made available to android phones. The project then achieved deployment by making the application available via the google play store in Kenya's country. A version was also made available in the United States for testing and debugging purposes. Figure 28 presents the infrastructure configured to stand up the ride-sharing platform.

Chapter VIII.

Testing

As part of the research for this Thesis, the initial study identified the interest of deploying a ride-sharing application on emerging markets shifting to accept cryptocurrencies such as Kenya. In the early phase of design for the thesis project, the research expanded to conversations with the drivers' consortium in Kenya. The driver's consortium's initial discussions resulted in the need and interest to get the application ready for real users to request rides in Kenya. The thesis project's testing phase played an essential role as a series of meetings were coordinated to get feedback from the driver's consortium and users in Kenya to get the application ready for real users. Figure 29 presents the application available for users in Kenya from the google play store for further reference.



Ride share application with cryptocurrency based payments via algorand platform. Provides a service for drivers and riders alike with live digital payment processing.

Figure 29. Rocket ride-share application on google play store

The Rocket ride-share application is available for users in Kenya via the google play store.

Some of the feedback from the driver's consortium was to adjust the button size. Nevertheless, even more importantly, some of the input is geared towards making the button text and flows as easily understood as possible.

Feedback provided also pointed out the importance of supporting package delivery as part of the options requested by a rider. The beta version includes the feature to support package delivery on the ride-sharing application released for testing. The restaurant market expects the emergence of digital restaurant concepts to continue, as operators recognize the upside of preparing food for takeout and delivery only. In addition to saving money on overhead and labor, ghost kitchens empower flexibility and creativity in menu innovation (Watrous, 2020).

The ride-sharing platform uses the google play console monitoring to track errors reported while testing the application. There has been only one error or crash detected by the google play console monitoring, as illustrated in Figure 30.

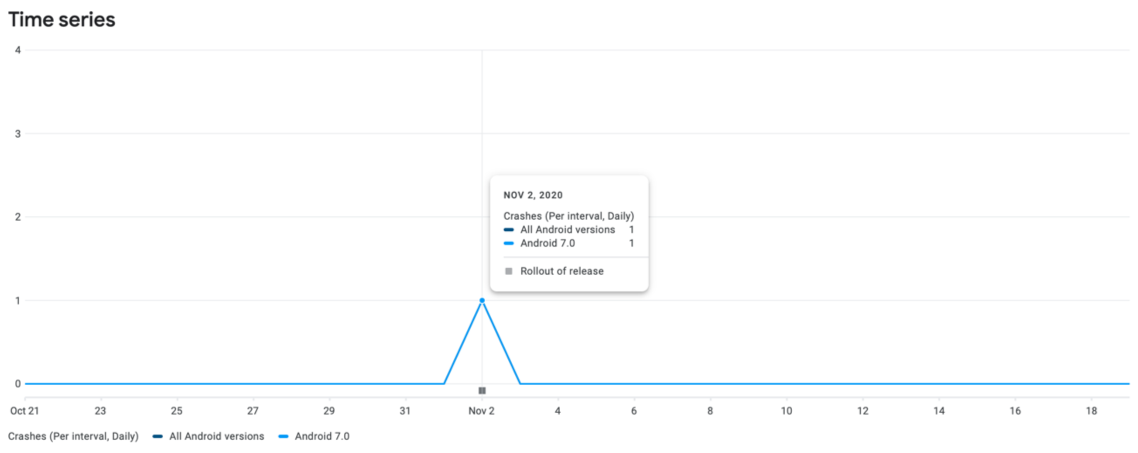


Figure 30. Ride-sharing app errors detected

Errors detected for the Rocket ride-share application up to November 18, 2020.

The google play console monitoring also provided the components to track new user registration trends. A slight increase in new user registration has been detected during October, as illustrated in Figure 31.

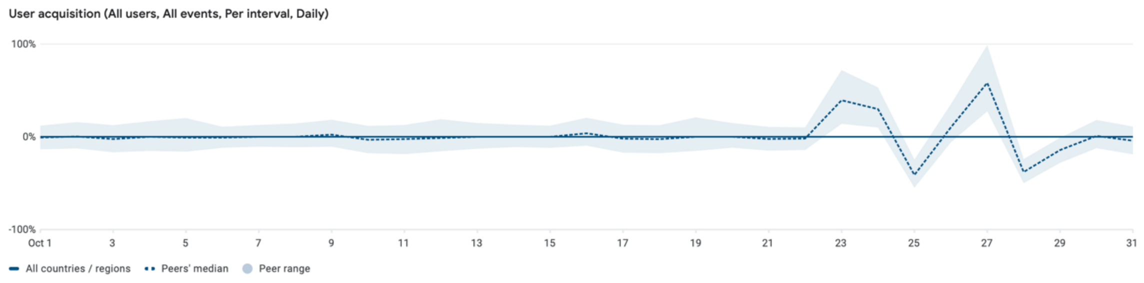


Figure 31. Ride-sharing app users registration monitoring

The Rocket ride-share user registration graph for October.

Chapter IX.

Future Work

As previously stated, the study initially proposed an approach to use Cadence workflow as the core of the orchestration for the ride-sharing platform as part of the trip management service. However, as the ride-sharing platform's further implementation took place, the benefits noted while using Cadence was apparent. However, they were more beneficial in a scope that required a workflow that involved more microservices in a given platform.

In Cadence, a workflow has full control over which activities an application executes and in which order. A workflow must not affect the external world directly, only through actions. What makes workflow code a workflow is that Cadence preserves its state. Therefore, any failure of a worker process that hosts the workflow code does not affect the workflow execution. The workflow continues as if these failures did not happen (Ahmad, 2020). Resiliency is a powerful and beneficial feature. However, the ride-sharing platform's current state comprises only three microservices. The orchestrator module can perfectly orchestrate the trip execution for those three microservices. If the ride-sharing platform is presented with the requirement to expand the platform to other regions or countries, this will require scaling up the platform. As a result, the number of microservices or host instances might increase, in which case, the benefit of using Cadence will be more significant to manage the workflow efficiently. If there is a considerable increase in the ride-sharing application users volume, the proper approach to scale up will be replacing the orchestrator module with Cadence.

A target area to improve is that of smart contracts and rewards for the ride-sharing platform. As mentioned in the design section, the wallet management service models a reward object to get the logic in place for a retribution model for both riders and drivers alike to reward using the ride-sharing platform. For the beta version released, the object right now is only a placeholder, and the rewards object adds no additional logic as part of the beta released to production.

Enter the Algorand blockchain, which supports defining assets (ASAs), representing fungible assets, including Algo, USDT, and USDC. Stateful TEAL is a smart contract language that allows maintaining global and local application state. A stateful TEAL application manages the state as part of the account information. The account linked to the application retains the global state in the crowdfunding application, including the fund configuration (goal, start date, end date, close date, and receiver account) and the total investment value. The local state associated with each investor's account tracks their investment (Gieseke, 2020).

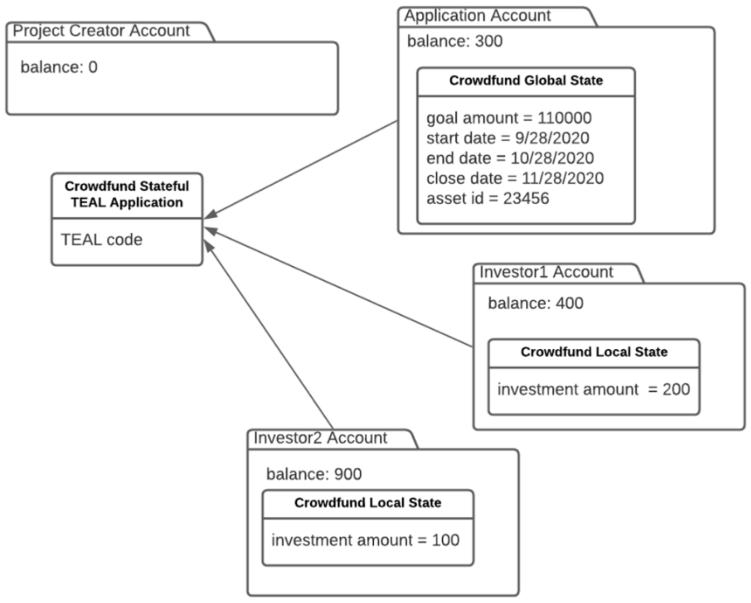


Figure 32. Crowdfunding TEAL stateful application components

The Crowdfunding TEAL stateful application components.

Figure 32 presents the crowdfunding TEAL stateful application components. Crowdfunding applications create a fund drive for a specific receiver that tries to reach a goal before a particular date. Individuals are allowed to donate to the fund up to a specific date. If the crowdfunding achieves the goal, the receiver can claim the funds. If not, the original givers can recover the donations (Weathersby, 2020).

The ride-sharing platform could benefit from a crowdfunding model; the future work identified is to use accreditation tokens to implement a reward system and replace the current smart contract module with crowdfunding stateful application components for the ride-sharing app.

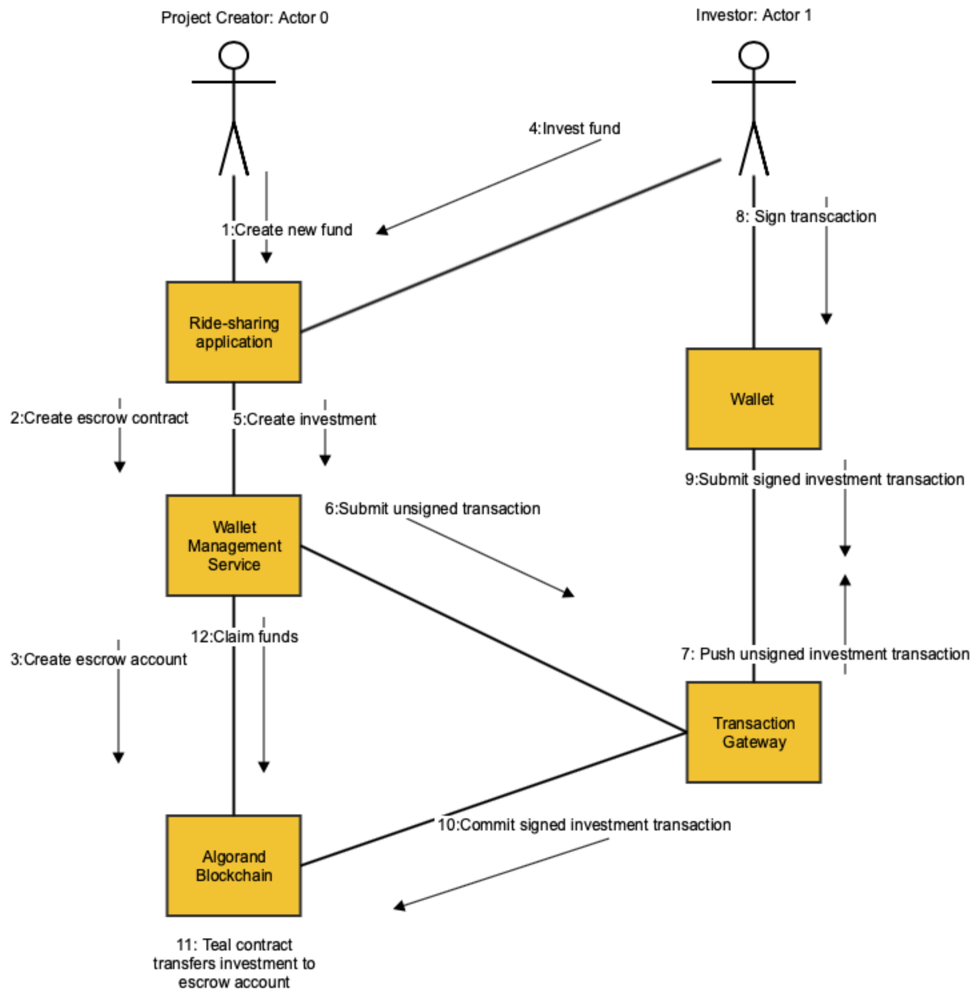


Figure 33. Crowdfunding workflow (Gieseke, 2020)

The proposed crowdfunding workflow for the ride-sharing application.

Figure 33 presents the proposed crowdfunding workflow to replace the smart contract module and implement rewards for the ride-sharing application. The enumeration below describes the workflow steps:

1. A project creator uses the android ride-sharing application to create a new crowdfunding project.

2. The android ride-sharing application calls the wallet management service to create a new crowdfunding project.
3. The wallet management service creates a new escrow account and associates it with the stateful TEAL application within the Algorand blockchain.
4. An investor uses the android ride-sharing application to invest in the fund.
5. The web android ride-sharing application calls the wallet management service to create an investment.
6. The wallet management service creates an investment transaction and sends it to the Transaction Gateway.
7. The Transaction Gateway notifies the signing wallet associated with the investment account that there is a transaction waiting to be signed.
8. The investor uses the signing wallet to inspect and then sign the transaction.
9. The signing wallet returns the signed transaction to the Transaction Gateway.
10. The Transaction Gateway submits the transaction to the Algorand blockchain.
11. The Stateful TEAL contract logic transfers the investment amount from the investor's account to the escrow account.
12. Finally, the project creator claims the funds accrued in the escrow account at the successful funding conclusion. The stateful TEAL contract transfers funds from the escrow account to the project creator's account.

Yet, one more target area to improve is the login process of the android ride-sharing application. As it stands on the current beta version deployed to Kenya, the android ride-sharing application prompts users to provide login credentials each time the application starts.

If it is not a banking application or something similar that stores sensitive user information, keeping users logged in might be an option. Bearing in mind that mobile users are usually on the go and with very little time or focus, it is quite helpful for users not to log out every time they turn off an app. Instead, the feature to keep the user logged in can be turned on (Levenson, 2020). However, it might be worth adding the features to offer remote session termination and or suggest a password change after a specific amount of time.

Chapter X.

Summary

One of this thesis's main objectives was to address current popular or mainstream security risks for both passengers and drivers. The implementation resulting from this thesis provides a viable alternative to managing those risks by introducing a blockchain-based solution to maintain records of decentralized and secure user data of both drivers and riders. The ride-sharing platform solution provides an alternative to address the security concern by allowing both riders and drivers alike to see other users' current ratings. Both riders and drivers can decide if they are to proceed with a given trip request after reviewing the other user's rating. Furthermore, both the rider and driver can rate each other after trip completion by validating users before the ride-sharing application provides the service as an option.

For this thesis's implementation, the history of ratings is exposed to other users to provide transparency and security while using the ride-sharing platform. However, the platform supports providing more data peer to peer as part of the user validation. Depending on the market, further thesis work can expand the study to discuss if the android ride-sharing application should make more data visible to users to proceed with a given trip request.

The solution implemented as a result of the thesis provides an alternative that improves the user's financial or economic profits over the most popular solutions. As mentioned during the abstract section, companies such as Uber and Lyft take on average a 20 percent cut of each ride transaction (Hmielewicz, 2018).

With the proposed solution, the benefit or gain from a ride transaction, the ride-sharing platform dramatically improves driver's profit by removing the third party or middle man that currently keeps 20 percent of the transaction. The ride-sharing platform eliminates the middleman factor by providing a ride-sharing platform based on blockchain technology as its core foundation. The solution provided enabled the opportunity to reduce the cost of transactions by eliminating the middleman via the Algorand blockchain platform to facilitate a direct transaction and secure payment transactions by interfacing with the Pago's transaction gateway.

Furthermore, the abstract section's research highlights that it could take a day to transfer money from one party to another. Specifically, the study's beginning identified that it could take days to transfer money for one party with an account in one country to another country (Swan, 2015).

The Algorand blockchain platform was the solution selected to improve the payment transaction processing throughput. The study evaluated Algorand's performance by standing up to 1,000 EC2 virtual machines and simulating up to 500,000 users to research scaling Byzantine Agreements for Cryptocurrencies. Experimental results showed that Algorand confirms transactions in under a minute, achieved 125 times Bitcoin's throughput, and incurs almost no penalty for scaling to more users (Gilad, 2020).

The ride-sharing platform dramatically enhances the processing time's performance for a payment transaction by reducing the time required to complete a payment transaction by securing a peer to peer transaction with no banks or any other institutions that might slow down the payment processing process. The payment of a given transaction using Pago's transaction gateway reduced the multi-day payment cycle down to real-time.

The thesis should highlight the package delivery feature that the ride-sharing platform implemented due to the study. The ride-sharing platform provides the service for user transportation and provides the components for delivering packages such as food or any other needs as a service. The ride-sharing platform provides such services via digital payments and is currently deployed to Kenya, an emerging market shifting to use or accept cryptocurrencies.

The thesis aimed to research the latest technologies and propose an implementation to enhance the ride-sharing industry's security and implement an application that will enhance ride-sharing costs for riders and profit drivers.

The ride-sharing application implemented as a result of this thesis attempts to enhance security in the ride-sharing industry. The mechanism used to rate peer to peer users and make that information available to the platform users is a viable solution to address security concerns. Nevertheless, the ride-sharing platform can enhance that mechanism by adding more data to the validation process. The platform can include data such as driver's license and registration number to the peer-to-peer validation mechanism. That will further improve the enhancement of security across the ride-sharing platform.

The cost of ride-sharing for riders and profit for drivers was significantly improved, as previously stated. With the solution provided as a result of this thesis, no middle-man or entity takes a percentage of the profit for a given payment transaction. Hence, the android ride-sharing application can provide more profit for drivers and more accessible fares for riders.

The performance for payment processing is a point that the study highlights due to its implementation. The ridesharing platform eliminates the multi-day payment cycle to process a given transaction and cuts down the processing time to real-time. The implementation achieves the performance leap by supporting the adoption of cryptocurrency in the payment industry. Furthermore, the implementation provides a reliable, viable solution in the ride-sharing sector that further cements the benefits of blockchain and cryptocurrency in emerging markets.

Appendix 1.

Glossary

API

API is the acronym for Application Programming Interface, the software intermediary used for applications to talk to each other.

Bitcoin

Bitcoin is a cryptocurrency invented in 2008 by an unknown person or group of people using the name Satoshi Nakamoto (Yellin, 2018).

Blockchain

A blockchain is essentially a distributed database of records or public ledger of all transactions or digital events that have been executed and shared among participating parties (Memon, 2018).

Cadence

Cadence is a workflow orchestrator. It is open-source, developed by Uber, and written in Go language (Ahmad, 2020).

Cassandra

Apache Cassandra is a database that focuses on reliable performance, speed, and scalability. It quickly stores massive amounts of incoming data and can handle hundreds of thousands of writes per second (Patel, 2020).

Decentralization

Decentralization is the transfer of responsibility for planning, management, and resource raising and allocation from the central government to field or subordinate units (Smith, 2001).

HTTP

HTTP stands for Hypertext Transfer Protocol. HTTP is the protocol used to transfer data over the web.

IoT

IoT or Internet of things is connecting any device with an on and off switch to the Internet. The devices can range from cell phones, coffee makers, washing machines, headphones, lamps, and wearable devices.

Maven

Apache Maven is a cornerstone of Java development and the most used build management tool for Java (Tyson, 2020).

MVC

The MVC or model view controller software architectural pattern separates concerns into three buckets (Doherty, 2020).

REST

REST is the acronym for Representational State Transfer. The architectural style for providing standards between computer systems on the web.

SOA

SOA or Service Oriented Architecture is a structure that allows services to communicate with each other across different platforms and languages by implementing what is known as a “loose coupling” system.

USDC

USD Coin (USDC) is the second-largest stable coin pegged to the US dollar in terms of market capitalization and usage. USDC launched in September 2018 as a collaborative effort between Circle and Coinbase. Like Tether (USDT), USDC is a major participant in the stablecoin market facilitating all sorts of money flow and use-cases.

References

Ahmad, J. (2020). *Cadence* — The only workflow orchestrator you will ever need, Noteworthy, Feb 23, 2020.

Agrawal, A. (2016). Five ways ridesharing is making the world a better place, Inc, Mar 06, 2016.

Alois, J. (2020). USDC Now on Algorand Bringing Major Scalability and Performance Improvements, Crowdfund Insider, Sep 09, 2020.

Ayoade, G., Karande, V., Khan, L., & Hamlen, K. (2018). Decentralized IoT Data Management Using BlockChain and Trusted Execution Environment.

Bambara, J., & Allen, P. (2018). Blockchain: A practical guide to developing business, law, and technology solutions. New York: McGraw-Hill Education.

Caballero, C. (2019). The Command Design Pattern. What is it and how can we apply it? Better Programming, Dec 05, 2019.

Carnell, J. (2017). Spring Microservices in Action (1st ed.). Manning Publications.

Dalul, S. (2019). What exactly is ridesharing and how do you get started as a driver? Android Authority, June 25, 2019.

Dudley, G., Banister, D., & Schwanen, T. (2017). The rise of Uber and regulating the disruptive innovator. The political quarterly, 88(3), 492-499.

Doherty, E. (2020). Educative: Interactive Courses for Software Developers. MVC Architecture in 5 minutes: a tutorial for beginners, Educative, May 10, 2020.

Fedak, V. (2018). Top 5 reasons to use the Apache Cassandra Database, Towards Data Science, Mar 02, 2018.

Gieseke, E. (2020). Creating a Point-of-Sale Application with the Algorand Blockchain, Algorand, Apr 13, 2020.

Gieseke, E. (2020). Creating a CrowdFunding Application with the Algorand Blockchain, Algorand, Oct 22, 2020.

Hwang, H. (2020). Measuring the Benefits of Ridesharing Services to Urban Travelers, Hutchins Center on Fiscal and Monetary policy at brookings, Oct, 2020.

Frankenfield, J. (2019). Cryptocurrency, Investopedia, May 05, 2020

Gilad, Y., Hemo, R., Micali, S., Vlachos, G. and Zeldovich, N. (2017). Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In Proceedings of the 26th Symposium on Operating Systems Principles.

Hahn, R. (2017). The Ridesharing Evolution: Economic Survey and Synthesis, Brookings, January 10, 2017.

Halaburda, H. (2019). Will we realize Blockchain's promise of decentralization? Harvard Business Review, Sep 04, 2019.

Hmielewicz, M. (2018). Food for thought Uber changes slow revenue growth. The Financial Times, p. 11.

Yli-Huumo, J., Ko, D., Choi, S., Park, S., & Smolander, K. (2016). Where is current research on blockchain technology?—a systematic review. PloS one, 11(10), e0163477.

Jesuthasan, R., Malcolm, T. and Cantrell, S. (2020). How the Coronavirus Crisis Is Redefining Jobs, Harvard Business Review, Apr 22, 2020.

Levenson, H. (2020). How to Perfect your Mobile App's Login Screen, Usability Geek, 2020.

Levi, S. (2018). An introduction to smart contracts and their potential and inherent limitations, Harvard Law School Forum on Corporate Governance, May 26, 2018.

Lewis, J. (2014). Microservices, a definition of this new architectural term, Martin Fowler, Mar 25, 2014.

McCarthy, A. (2020). The Rise of Food Delivery Services: Why Today's Consumers Order In, Doordash for Merchants, May 22, 2020.

Memon, M., Hussain, S. S., Bajwa, U. A., & Ikhlas, A. (2018, August). Blockchain beyond bitcoin: Blockchain technology challenges and real-world applications. In 2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE) (pp. 29-34). IEEE.

Michael. (2020). Algorand (ALGO) Explained, Boxmining, Nov 21, 2020.

Mire, S. (2018). Blockchain in Ridesharing: 5 Possible Use Cases. Disruptor Daily.

Sonatype Company Staff Corporate Author. (2008). Maven: The definitive guide (First ed.). Sebastopol, California: O'Reilly.

O'Neal. (2019). Cointelegraph. (n.d.). Who Scales It Best? Inside Blockchains' Ongoing Transactions-Per-Second Race. Cointelegraph.

Russon, M.A. (2019). Crypto-currencies gaining popularity in Kenya, BBC News, Feb 22, 2019.

Sonatype Company Staff Corporate Author. (2008). Maven: The definitive guide (First ed.). Sebastopol, California: O'Reilly.

Patel, N. (2020). What is Cassandra and why are big tech companies using it? Canonical, May 05, 2020.

Xu, R., Ramachandran, G. S., Chen, Y., & Krishnamachari, B. (2019, October). Blendsm-ddm: Blockchain-enabled secure microservices for decentralized data marketplaces. In 2019 IEEE International Smart Cities Conference (ISC2) (pp. 14-17). IEEE.

Schaut, S. (2020). What is a chargeback? Credit karma, December 14, 2020.

Saleem, S. (2019). What Is DigitalOcean and Why Should You Select It for Your Web Hosting? The Official Cloudways Blog, Mar 27, 2019.

Serrano, S. (2020). Saving (in) Latin America: Why stable coins are thriving across emerging markets, Circle Blog, Oct 14, 2020.

Smith, L. (2001). Reform and Decentralization of Agricultural Services: A Policy Framework, p, FAO Agricultural Policy and Economic Development Series, 2001.

Cynthia, S. T., Majumder, M., Tabassum, A., Khanom, N. N., Tuhin, R. A., & Das, A. K. (2019, September). Security Concerns of Ridesharing Services in Bangladesh. In 2019 2nd International Conference on Applied Information Technology and Innovation (ICAITI) (pp. 44-50). IEEE.

Suryotrisongko, H., Jayanto, D., & Tjahyanto, A. (2017). Design and Development of Backend Application for Public Complaint Systems Using Microservice Spring Boot. Procedia Computer Science, 124, 736-743.

Swan, M. (2015). Blockchain: Blueprint for a new economy (First ed.). Sebastopol, California: O'ReillyMedia.

Thomson, J. (2016). Uber driver assaulted in Miami: How safe is the system? The Christian Science Monitor, The Christian Science Monitor, Jan 22, 2016.

Tyson, M. (2020). What is Maven? Build and dependency management for Java, Info World, Jan 30, 2020.

Yellin, T., Aratari, D., Pagliery, G. (2018). What is bitcoin? CNN Money, Aug 08, 2018.

Yowana, W. (2020). Credit card chargebacks explained: A guide for merchants, Valuepenguin, Aug 17, 2020.

Watts, S. (2017). What is service-oriented Architecture? BMC Software, May 31, 2017.

Walters, S. (2019). Algorand Review: ALGO Worth It? Everything You need to Know, Coin Bureau, Sep 08, 2019.

Watrous, M. (2020). Ten trends emerging in the aftermath of COVID-19, Food Business Review, Nov 18, 2020.

Weathersby, J. (2020). Example Crowdfunding Stateful Smart Contract Application, Algorand, Aug 20, 2020.

Wittmer, P. (2020). Monolithic vs Microservices Architecture - Why Microservices Win, Tiempo Development, Mar 12, 2020.