<< Back

1. Introduction and Goals

1.1 Requirements Overview

Use Case Diagram

User Registration and Authentication:

Driving Forces: Ensure security and accountability. Create personalized user experiences.

Profile Management:

Driving Forces: Allow users to maintain and update personal information. Enable customization of preferences and settings.

Ride Booking:

Driving Forces: Provide a quick and user-friendly process for requesting rides. Offer features such as real-time location tracking and estimated arrival times.

Driver / User Matching:

Driving Forces: Implement algorithms for efficient driver-passenger matching. Consider factors such as proximity, driver rating, and user preferences.

Price Calculation and Payment:

Driving Forces: Provide transparent and fair pricing. Support various payment methods for user convenience. Price calculation based on distance and current demand.

Route Optimization:

Driving Forces: Optimize routes to ensure timely and efficient journeys. Consider factors like traffic, road conditions, and user preferences.

Real-time Tracking:

Driving Forces: Enhance user safety and confidence. Enable users to monitor the location of their ride in real-time.

1.2 Quality goals

Content

The top three quality goals for the architecture whose fulfillment is of highest importance to the major stakeholders considering the ISO 25010 standard.

Priority	Quality	Motivation
1	Usability	

Priority	Quality	Motivation
		User-friendly experience leads to higher user counts
2	Security	Payment services handle sensitive information, therefore a safe environment is required.
3	Availability	Easy scalability for peak times, therefore efficient use of infrastructure resources for cost reduction.

Motivation

In the ride-sharing app context, addressing usability, security, and availability aligns with the goal of providing a convenient, safe, and reliable transportation service. The app becomes not only a functional tool for connecting passengers with drivers but also a user-friendly and trustworthy platform that meets the diverse needs of its users.

1.3 Stakeholder

The table below provides an overview of all stakeholders involved in the M.O.T.U.S. project.

Name	Role	Expectations
Maryam Patel	Investor at Venture Capital Firm XYZ	Profit, high return of investment
Raj Gupta	Business Development Manager at Local Transportation Authority	Expects the startup to contribute positively to the local transportation business landscape
Megan Chen	User Experience Designer at	Expects the startup to provide a good user experience, and

Name	Role	Expectations
	Creative Agency ABC	wants to include this success in her agency’s portfolio
Amirah Rahman	Environmental Activist and Representative of Sustainable Transportation NGOC	Advocating for eco-friendly and sustainable transportation solutions to mitigate environmental impact and promote a greener, more sustainable future
Javier Gomez	RideShare Driver and Representative of RideShare Driver Association	Expects the startup to treat workers fairly and cooperate with the RideShare Driver Association
Developers	Design, develop and maintain App	Clear and refined requirements, environment where developers can focus on their work, deliver high-quality software

<< Back

Architecture Constraints

- MOTUS will require a reliable and stable internet connection to function properly.
- MOTUS will require location services on end-user devices.
- MOTUS is a mobile-first application; that means it is designed as an app and not as a website.
- MOTUS will run on Android and iOS devices natively.
- MOTUS must respect data protection and privacy laws.
- MOTUS will be subject to local regulations and legal requirements, which may impact its availability and functionality in certain areas.
- MOTUS will require ongoing maintenance and support to ensure optimal performance and user experience.
- MOTUS will need to compete with existing ride-sharing and transportation services in the market.

<< Back

Scope and Context

Scope

The app will be available for both iOS and Android devices.

The app will allow users to create profiles, search for rides, and connect with nearby drivers.

The app will include a payment system to facilitate transactions between drivers and passengers.

The app will use location-based services to identify nearby drivers and passengers.

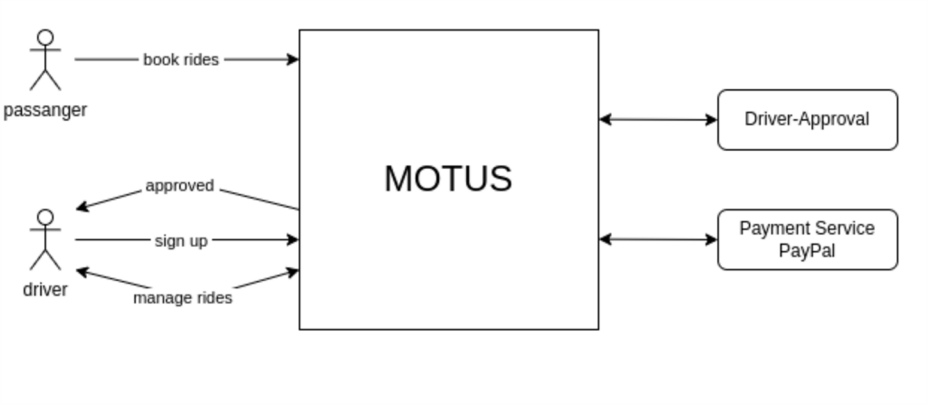
The app will provide real-time updates on ride availability and estimated arrival times.

Business Context

List of stakeholders:

- Passengers (want the ride)
- Drivers (provide the ride)
- Approval center (approve drivers)
- Payment service

Representation



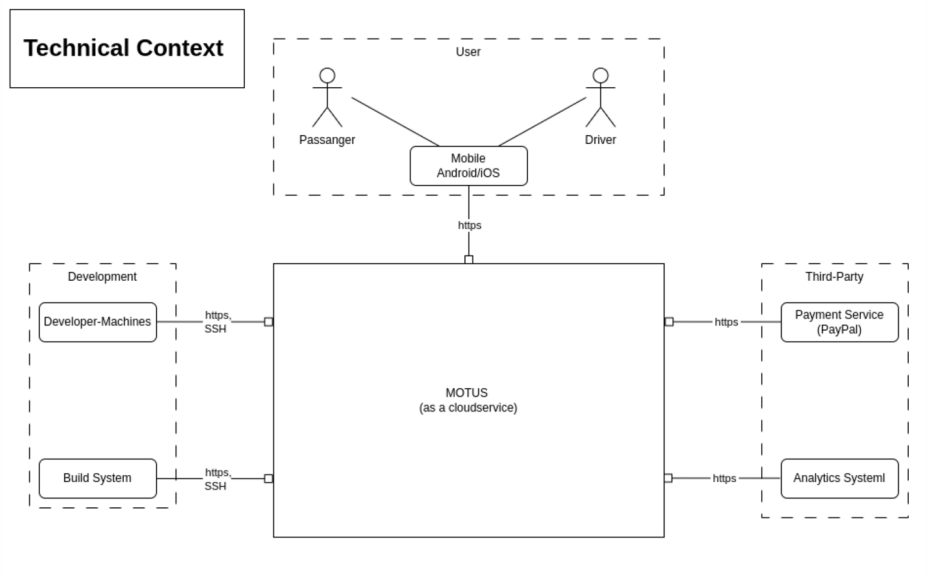
A simple business diagram

Piece	Purpose
MOTUS	Our service to connect passengers and driver plus a payment service.
Approval Center	Legal approval if drivers are allowed to drive. Receive a form and contact details of drivers via web-interface.
Payment Service	Handles payment of users and payout for drivers. Integrated over API with redirect to the payment provider.
Passenger	User can sign up to use our app.
Driver	Provides driving service to passenger.

Technical Context

List of stakeholders: - App users - Developer - Approval center (approve drivers) - Payment service - Analytics service

Representation



A diagram of the technical context

Piece	Purpose
MOTUS	Our service to connect passengers and driver plus a payment service.
Approval Center	Legal approve if drivers are allowed to drive. Receive a form and contact details of drivers via web-interface.
Analytics service	A third party service to monitor the cloud and services running on it.
Payment Service	Handles payment of users and payout for drivers. Integrated over API with redirect to the payment provider.
App user	Connect to our services over the internet from Android/iOS devices.
Developer	Human actor, responsible for App development, deployment and maintenance.

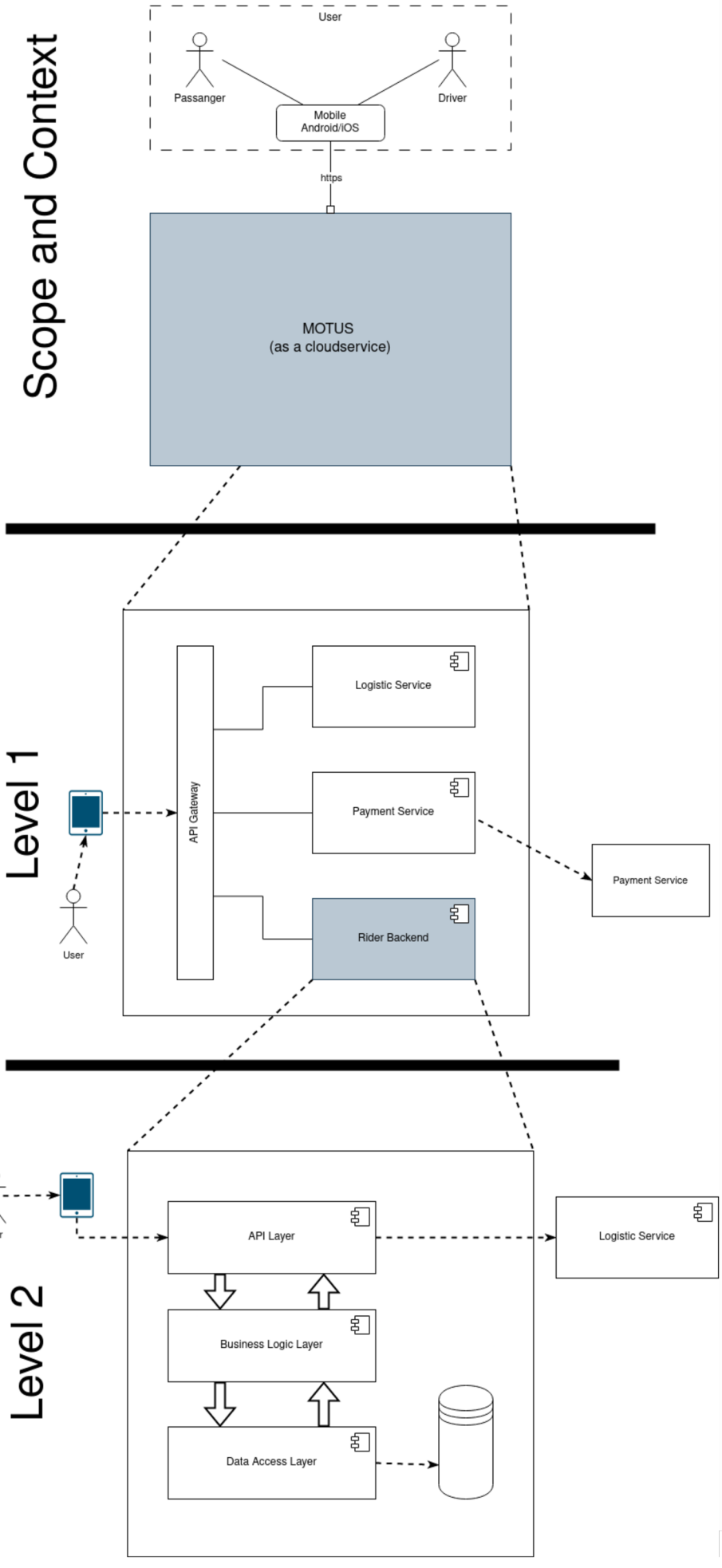
<< Back

Solution Strategy

1. The server application is written in Java.
    - Our core developer team has Java experience.
    - We will use the industry-standard Spring framework.
2. The clients are written in Flutter, which streamlines development for Android and iOS.
    - We have some challenges in this area: we do not have any in-house Swift experience.
    - Native code implies some doubled work.
    - The user experience must be seamless, which requires native code.
3. The client applications have feature parity with each other.
4. The clients communicate with the server via a well-defined API.
    - Flutter will help us to streamline our development process into a single codebase, in order to reduce code overhead and redundancy.
5. The system is designed as microservices.
    - We anticipate that there will be higher initial costs, but as our start-up expands, this architecture will enable individual teams to own and manage specific microservices.
6. The system is fully containerized to improve scalability, portability, and service isolation.
    - To address the issue of peak times, we adopt a microservices architecture for scalability.
7. The system is resilient; we are dependent on growth and must therefore plan for downtime solutions.

<< Back

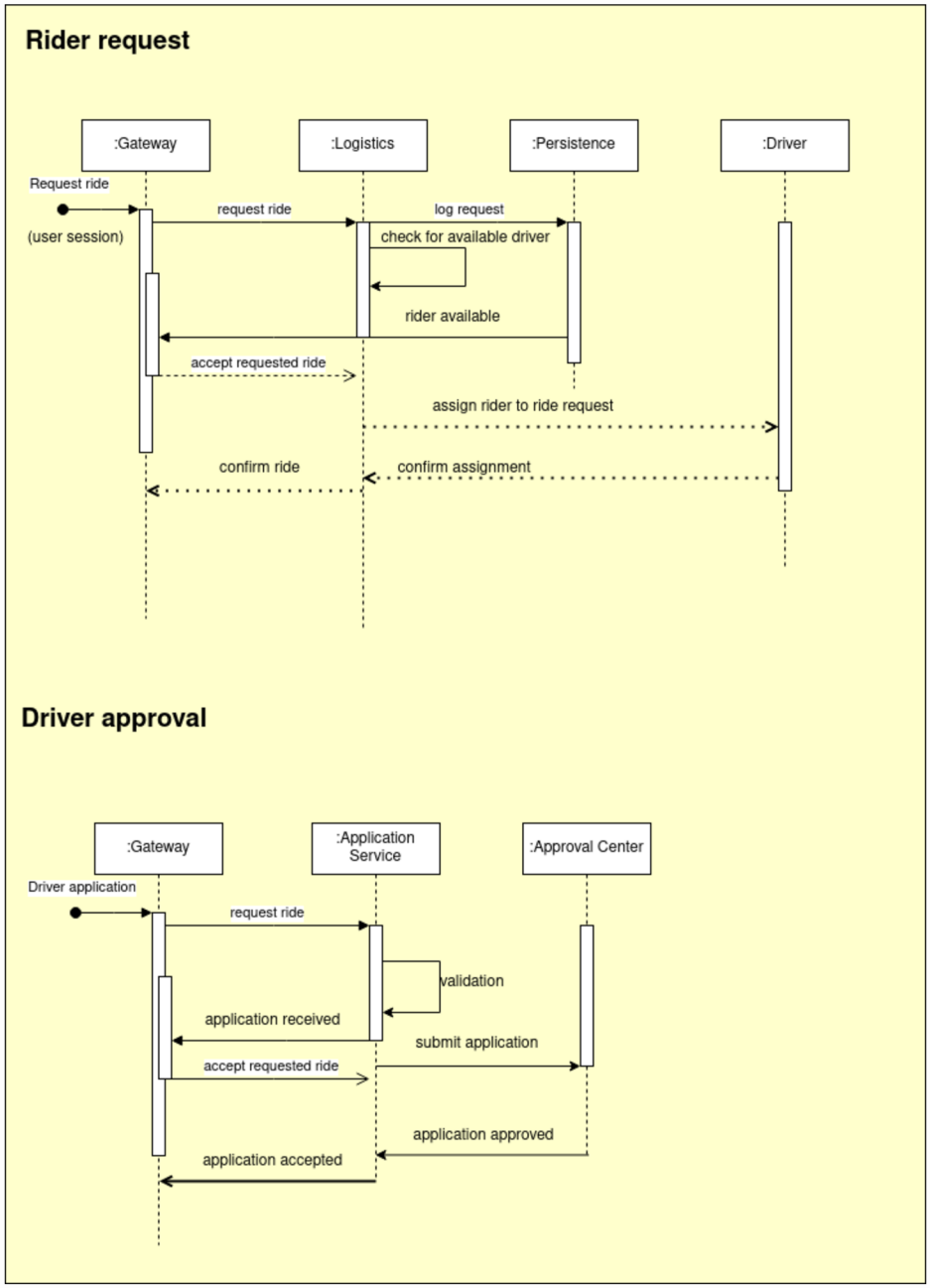
Building Block View


Scope and Context
User
Passanger
Driver
Mobile
Android/iOS
https
MOTUS
(as a cloudservice)
Level 1
User
API Gateway
Logistic Service
Payment Service
Rider Backend
Payment Service
Level 2
User
API Layer
Logistic Service
Business Logic Layer
Data Access Layer

Building Block View

<< Back

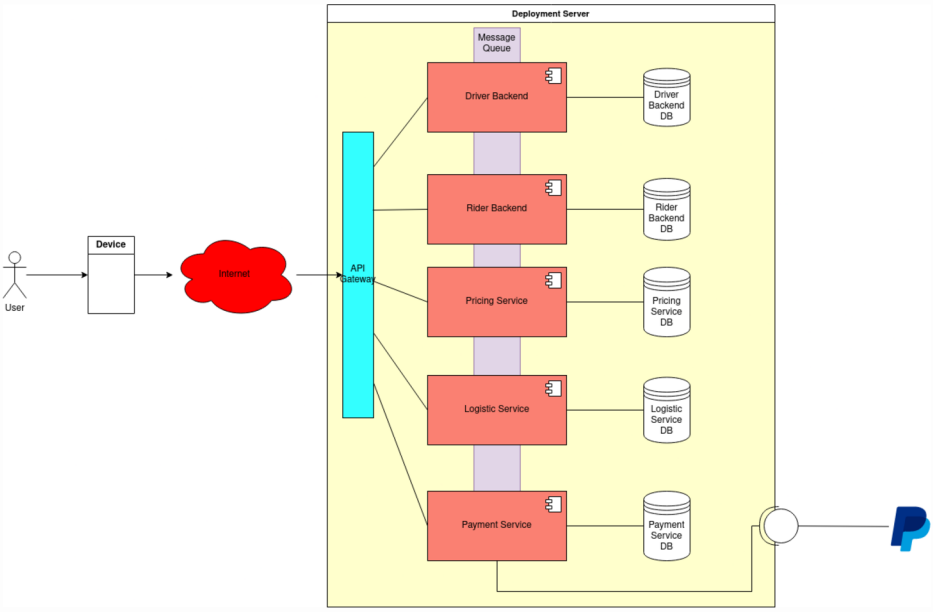
Runtime view



Runtime view

<< Back

Deployment View



Deployment View Diagram

Node/Artifact	Description
Load Balancer	Multiplexes client requests to separate instances of the entire VPC.
API Gateway	Used for security, request aggregation, quality assurance, routing, etc.
MOTUS Driver Backend	REST server for the driver-facing API
MOTUS Passenger Backend	REST server for the passenger-facing API
MOTUS Logistic Service	Responsible for calculating routes and available drivers.
MOTUS Pricing Service	Responsible for calculating trip fare.

Node/Artifact	Description
MOTUS Payment Interface	Backend microservice for communicating with the external payment service.
MOTUS Message Queue	Inter-Service Communication, i.e. RabbitMQ
MOTUS Logging Interface	Provides a centralized interface for logging, i.e. ElasticSearch/DataDog etc.

<< Back

Cross-cutting Concepts

Architecture and Design Patterns

1. Microservice architecture
2. Design Patterns
    - SOLID Principles
    - Factory pattern, singleton classes, etc.
3. Domain-driven design
    - The design of our architecture must represent real-world entities such as passengers, drivers, rides, payments, etc. This must be consistent across the entire application.

2. Security & Safety

1. Security
    - Data
    - Humans
2. Safety
    - Data
    - Humans
3. Data consistency / eventual consistency
    - We are reliant on growth, so we need to have rules and patterns for consistency that are clear and coherent while relying on eventual consistency for its scalability.
4. Resilience and fault tolerance

3. Development Concepts

1. Test Driven Development
2. Continuous Integration / Continuous Deployment

3. Performance as a first principle
    - The application must work at scale. This is critical to our survival as a business.
4. Configurability

<< Back

Microservices over Monolith

- Context
  Microservices are more future-proof than monolithic applications.
- Decision
  We decided to build microservices in order to maximise the scalability of our application. We are dependent on growth in order to succeed, and our services therefore need to be flexible and scalable.
- Status
  The stakeholders agreed with the plan; they agree that we need to be as future-proof and scalable as possible.
- Consequences
  Our core developer team is still relatively small; we're trading initial overhead for eventual scalability.

PayPal integration via library

- Context
  We don't have capacity at the moment to maintain an in-house payment system.
- Decision
  We decided to use the well-established PayPal integration library "Super Library" for this purpose instead of reinventing the wheel.

- Status
  Accepted after a demonstration showed that the implementation is correct for our use case.
- Consequences
  We cannot control the details of the implementation.

Spring Boot as our Backend Technology

- Context
  We have in-house developer experience
- Decision
  Because of the existing resource of developer experience and the widespread usage of Spring Boot, we decided to build our backend with this framework.
- Status
  Accepted after a meeting with the stakeholders; they were of the opinion that using our existing experiences was the correct approach.
- Consequences
  If any problems or incompatibilities between other services and technologies appear, we will have a hard time reimplementing the backend.

Amazon CloudWatch as Analytics Software

- Context
  Our cloud provider is AWS and has a vast product variety. Used for monitoring resource usage.

- Decision
  For ease of use and fast integration, we stay with Amazon since they are also our cloud provider.

- Status
  Not fully accepted. Approval missing in reagard of costs and neccessity. There is demand from the development team.

- Consequence
  Adds another layer of tasks for our development team.
  Increase of costs.
  We are even more dependent on a single provider.
  Helps gather statistics and numbers not only for developer but also for business partners.

<< Back

Quality Attributes (ISO 9126)

- Portability
  - Replace-ability
    - Components are designed such that they can be replaced with new technologies if needed.
  - Installability
    - The services are constructed to be deployed quickly and simply whenever new features are completed.
- Maintainability
  - Stability
    - Normal use of the services does not cause crashes or loss of data.
  - Testability
    - Features have concrete interfaces which are thoroughly tested.
- Efficiency
  - Time behaviour
    - User-facing database transactions return within 500ms.
  - Resource demand
    - Inactive instances do not consume machine resources, thereby saving on cloud infrastructure costs.
- Functionality
  - Security
  - Accuracy
- Reliability
  - Recoverability
    - The system is able to be restored to a recent state.
- Usability
  - Learnability
    - Tips are shown to the user
  - Understandability
    - Interfaces for users are easy to understand, precise and concise.

Quality Scenarios

1. Driver accepts a contract. A canceled request is shown in a span of 2-3 seconds.
2. User opens the application. All possible routes are shown in under 5 seconds.
3. The user makes a request that fails. The user is informed about the error and its reason. The error is shown in under 10 seconds.

<< Back

Technical Debt

Performance Optimization

Reason

In reducing code complexity and fast scaling, storage optimization is not a top priority.

Reduction efforts

- Reducing non critical redundancy.

Risks

Vendor Lock-in

Reason

In reducing complexity and cutting cost at the team, only one cloud provider is chosen. Self hosting is not an option. Only one expert in cloud computing is needed to build our product.

Mitigation efforts

- Keep a clear structure of our services.
- Do stuff in house rather than using all services of the cloud provider.

Version Issues

Reason

Complex versioning arises in the microservice architecture. All services need to stay compatible with each other and on many different devices.

Mitigation Efforts

- Strict adherence to versioning conventions.
- Centralized version tracking.

<< Back

Glossary

Term	Definition
Ride	A unit of work for our application.
User	Either a rider or a driver.
Rider	A user who wants to be driven somewhere; pays for the privilege and expects good service.
Driver	A user who drives riders to destinations, in order to earn money.
Client	A program running on a user’s device; drivers and riders have separate client applications.
Microservice	A standable application that fits into a larger whole; has a single responsibility that it must correctly fulfill.
API	Application Programming Interface: provides a well-defined method of communication between clients and microservices.
Scalability	One of our core design decisions; we will only succeed if we grow, and to grow we must consider scalability as a first-principle.