Integrated Public Transportation System Project Proposal

Dharini Baskaran*
Surya Rajendran*
dharini.baskaran@colorado.edu
surya.rajendran@colorado.edu
University of Colorado Boulder
Boulder, Colorado, USA

1 INTRODUCTION AND MOTIVATION

In the evolving landscape of urban mobility, the need for seamless and efficient public transportation systems has become paramount. This project proposes the development and implementation of an Integrated Public Transport System—a pioneering initiative designed to unify various modes of public transportation into a cohesive and user-friendly framework. This system aims to streamline the travel experience for commuters by allowing them to make a single payment that covers multiple transportation modes within a 90-minute window.

In a city like Boulder, public transportation offer users a range of options, including Day Passes, RTD MyRide cards, 3-hour Cash Passes for bus transport, participation in the Bcycle program, utilization of Nightrides, engagement in RideShares, and more. However, the existing challenge lies in the fragmented nature of these services, where users are compelled to manage separate passes or applications to monitor their usage and make payments across various transportation modes. Our proposed project aims to introduce a unified payment solution—a singular card or payment mode—that seamlessly transcends all transport zones. This comprehensive system is designed not only to simplify the user experience but also to incentivise regular use of public transportation. By consolidating disparate modes of transport into a singular, user-friendly platform, our initiative seeks to encourage widespread adoption of public transit, offering an integrated and efficient solution that promotes the ease and attractiveness of utilizing the region's diverse transportation services.

The Integrated Public Transport System addresses this challenge by leveraging data scaling techniques to seamlessly integrate disparate modes of transportation. By employing advanced techniques, the system will facilitate a unified payment process, allowing users to pay for their entire journey with a single transaction. This innovative approach not only enhances user convenience but also has the potential to significantly increase public transport ridership.

Moreover, the project aligns with broader sustainability goals by contributing to the reduction of carbon emissions in urban environments. Encouraging the use of public transportation over individual vehicles is a key strategy in mitigating the environmental impact of urban commuting. Through the seamless integration of payment systems, the project aims to create a more attractive and user-centric public transportation experience, ultimately promoting eco-friendly modes of travel.

2 SOFTWARE COMPONENTS

(1) **SQL:** We will be using the relational tables and database predominantly for our project. It will be used for RFID tag management, Fare deduction services, payment deduction and top-up services and for recording every user's journey in a day.
(2) **NoSQL:** We will be incorporating NoSQL database for our Transactions database. This transaction table will be handling details of payments, time-expiration and so-on.
(3) **Redis MQ:** Redis MQ as a messaging system will be used in our project for asynchronous events for our EDD (Event-Driven-Design). It will facilitate real-time updates in our system.
(4) **Redis Cache:** Redis Cache will be utilized to enhance our system performance by storing data in-memory. We can share frequently used data like busId, fare rate for different zones and more.
(5) **Containers:** We will be using Docker containers for easy scalability of our public system. Containers will play an important role in achieving a modular architecture for our project. This ensures that the system will provide reliability. Kubernetes will be used to orchestrate these containers.

3 ARCHITECTURAL DIAGRAM

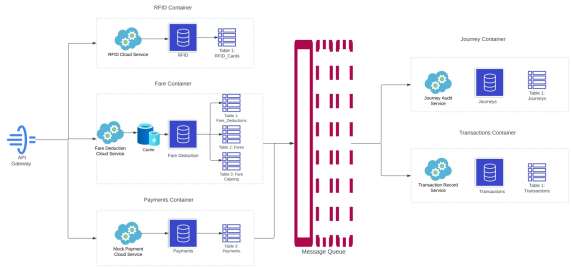


Figure 1: Architecture Diagram

*Both the authors contributed equally to this work.

4 SOFTWARE INTERACTION

In the realm of backend software development, seamless interaction between various services is pivotal for a robust and responsive system. This section elucidates the software interactions within our integrated public transport system's service backend system, focusing on two primary endpoints: the Tag endpoint and the Top-up endpoint, both of which are crucial for the operation of a public transport card usage system.

The Tag endpoint is the first point of contact for the user when they employ their transport card across different services such as buses, trains, metros, and bicycles. Upon the use of a transport card, a request is initiated which is intercepted by an API Gateway. The Gateway serves as a conduit to three distinct cloud services, each specialized for a specific function - RFID, Fare, and Payments. These services are not standalone; they each interact with dedicated databases to access and store pertinent information. The RFID service takes precedence in the process flow, its primary task being the validation of the transport card through the retrieval of the RFID tag. A valid card triggers the next phase where the system acquires the geolocation of the transport, thereby determining the mode of transportation utilized by the user.

Subsequently, the execution progresses to ascertain the fare. This entails a cache request for the zone information. If the cache lacks this data, a database query is made. The fare calculation is dynamic, taking into account the zone, mode of transport, and the user's category, such as veteran, student, or senior citizen. Furthermore, to promote the use of public transport, a fare-capping feature is incorporated. This feature ensures that beyond a certain threshold of usage, discounted rates are applied to journeys, effectively capping the fare to encourage frequent use.

After the fare deduction, a response is promptly dispatched to the user. Concurrently, an event is created and passed through a Message Queue to the Journey Audit Service. This service is responsible for the assimilation of audit data within the database, aiding in analytics and performance assessments. The audit data generation is performed asynchronously, allowing for the simultaneous operation of sending the user response and creating audit events to occur parallelly

Turning attention to the Payment endpoint, it is activated when a user opts to recharge their transport card. The process initiates with a call to the Payment Service, which in turn interfaces with a payment gateway, Stripe API being the selected gateway for this project's scope. Upon successful completion of the payment, the transaction details are recorded in a transaction database table. The Payment Service generates an event that is dispatched to Redis, acting as an intermediary message broker. The Transaction Record Service picks up this event from Redis to maintain transaction records. This operation is also executed asynchronously to ensure efficiency. Once the transaction record is created and confirmed as successful, the details are relayed back to the user. Redis, once again, plays a pivotal role by serving as the message queue from which the Email Service retrieves the transaction details to notify the user.

This document has detailed the intricate interactions between software components that underpin the backend of a transport service system, highlighting the seamless integration of cloud services, databases, caching mechanisms, and asynchronous processing to deliver a sophisticated and user-centric service.

5 DEBUG

Here, we propose leveraging Datadog as a robust tool for debugging and monitoring our system. Datadog offers comprehensive logging capabilities that enable us to gain deep insights into the inner workings of our application. By strategically implementing loggers within our codebase, we can track critical events, errors, and performance metrics. Datadog's intuitive dashboard allows us to visualize and analyze these logs in real-time, facilitating efficient debugging and issue resolution. This approach ensures that our development team can swiftly identify and address any anomalies or errors in the system, enhancing the overall reliability and stability of our project.

In addition to its logging capabilities, Datadog serves as a powerful tool for monitoring the status of our cloud services. By integrating Datadog with our cloud infrastructure, we can receive real-time updates on the health and performance of various cloud components. Datadog's comprehensive suite of integrations allow us to track metrics such as server response times, resource utilization, and network latency. With proactive monitoring, our team can anticipate potential issues before they escalate, ensuring optimal system performance and minimizing downtime. Datadog's centralized platform provides a holistic view of our cloud environment, empowering us to make data-driven decisions for continuous improvement and ensuring a seamless user experience for our project stakeholders. Addition to Datadog, we will be performing unit tests for our components.

6 INSPIRATIONS

(1) MARTA - Metropolitan Atlanta Rapid Transit Authority, Atlanta.
(2) RTD - Regional Transportation District, Denver.
(3) MTA - Metropolitan Transportation Authority, New York City.
(4) BART - Bay Area Rapid Transport, San Francisco.
(5) MBTA - Massachusetts Bay Transport Authority, Boston.

7 APPENDIX

Attaching Database Schema and API details.

Appendix 1 : Database Schema

Service: RFID Card Management Service

RFID_Cards Table:

Column Name	Data Type	Constraints
Card_ID (PK)	VARCHAR(16)	Primary Key, Not Null
User_ID	INT	Foreign Key (Users), Nullable
Balance	DECIMAL(10,2)	Not Null
IsActive	BOOLEAN	Not Null, Default True
Created_At	TIMESTAMP	Default Current Timestamp
Updated_At	TIMESTAMP	Default Current Timestamp

Service: Fare Deduction Service

Fare_Deductions Table:

Column Name	Data Type	Constraints
Deduction_ID (PK)	INT	Primary Key, Not Null
Card_ID (FK)	VARCHAR(16)	Foreign Key (RFID_Cards), Not Null
Journey_ID (FK)	INT	Foreign Key (Journeys), Not Null
Amount	DECIMAL(10,2)	Not Null
Tagged_On_Timestamp	TIMESTAMP	Not Null
Expiration_Time	TIMESTAMP	Not Null
Created_At	TIMESTAMP	Default Current Timestamp
Updated_At	TIMESTAMP	Default Current Timestamp

Fares Table:

Column Name	Data Type	Constraints
Fare_ID (PK)	INT	Primary Key, Not Null
Mode_of_Transport	VARCHAR(50)	Not Null
Zone	INT	Not Null
Amount	DECIMAL(10,2)	Not Null

Fare_Capping Table:

Column Name	Data Type	Constraints
Fare_Capping_ID (PK)	INT	Primary Key, Not Null
Mode_of_Transport	VARCHAR(50)	Not Null
Zone	INT	Not Null
Time_Period	INT	Not Null
Max_Amount	DECIMAL(10,2)	Not Null

Service: Journey Audit Service

Journeys Table:

Column Name	Data Type	Constraints
Journey_ID (PK)	INT	Primary Key, Not Null
Card_ID (FK)	VARCHAR(16)	Foreign Key (RFID_Cards), Not Null
Mode_of_Transport	VARCHAR(50)	Not Null
Start_Time	TIMESTAMP	Not Null
End_Time	TIMESTAMP	Nullable
Fare_Deducted	DECIMAL(10,2)	Not Null, Default 0.00
Tagging_Status	VARCHAR(10)	Not Null
Created_At	TIMESTAMP	Default Current Timestamp
Updated_At	TIMESTAMP	Default Current Timestamp

Service: Transaction Record Service

Transactions Table:

Column Name	Data Type	Constraints
Transaction_ID (PK)	INT	Primary Key, Not Null
Card_ID (FK)	VARCHAR(16)	Foreign Key (RFID_Cards), Not Null
Amount	DECIMAL(10,2)	Not Null
Transaction_Type	VARCHAR(50)	Not Null
Timestamp	TIMESTAMP	Not Null
Created_At	TIMESTAMP	Default Current Timestamp
Updated_At	TIMESTAMP	Default Current Timestamp

Service: Payment Service

Payments Table:

Column Name	Data Type	Constraints
Payment_ID (PK)	INT	Primary Key, Not Null
Card_ID (FK)	VARCHAR(16)	Foreign Key (RFID_Cards), Not Null
Amount	DECIMAL(10,2)	Not Null
Payment_Method	VARCHAR(50)	Not Null
Status	VARCHAR(20)	Not Null
Created_At	TIMESTAMP	Default Current Timestamp
Updated_At	TIMESTAMP	Default Current Timestamp

Appendix 2: API Gateway

RFID Card Management API

Overview

The RFID Card Management API provides functionality for handling RFID card interactions, specifically for recording user entry and exit onto modes of transport. This API is designed to be integrated with an RFID card reader system.

Base URL

```
https://api.example.com/rfid
```

Endpoints

1. Tag Endpoint

Description

Records the entry or exit of a user onto a mode of transport when they tap their RFID card. If the user has not started a journey, it tags ON. If the user is already in a journey, it tags OFF. If the reader is in a different mode of transport, it also tags ON.

Endpoint URL

```
POST /api/tag
```

Request

- **Card ID**: Unique identifier of the RFID card (string or integer).
- **Journey Status**: Boolean indicating if the user is already in a journey (True) or not (False).
- **Reader Mode**: Mode of transport of the reader (string).

```
{
"card_id": "100020000190",
"journey_status": false,
"reader_mode": "bus"
}
```

Response

- **Success**: Boolean indicating if the tagging process was successful.

```
{
"success": true
}
```

2. Top-up Endpoint

Description

Allows users to add value to their RFID card, increasing the card's balance for fare payments.

Endpoint URL

```
POST /api/topup
```

Request

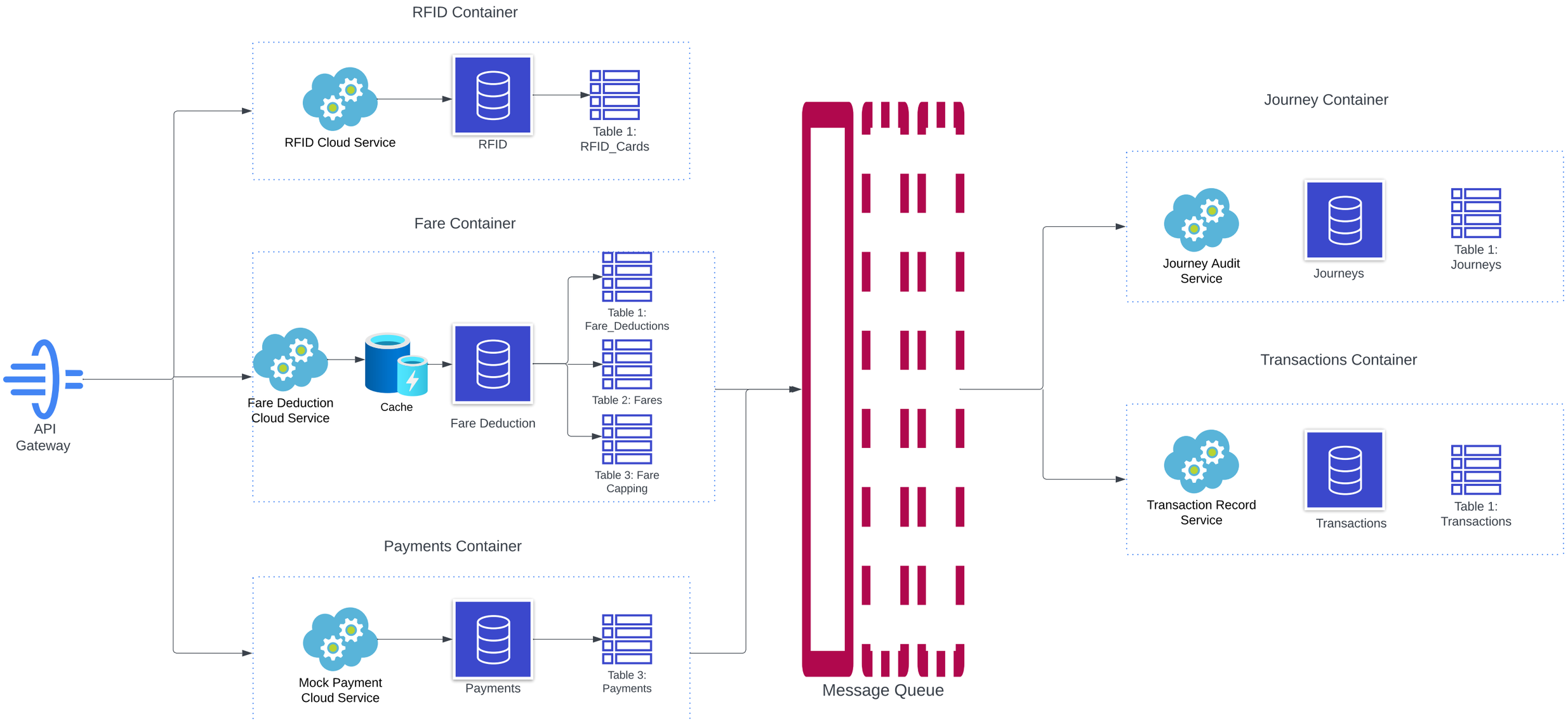
- **Card ID**: Unique identifier of the RFID card (string or integer).
- **Amount**: Amount to top up (float or integer).

```
{
"card_id": "100020000190",
"amount": 20.00
}
```

Response:

- **New Balance**: New card balance after top-up

```
{
"new_balance": 35.00
}
```


RFID Container
RFID Cloud Service
RFID
Table 1:
RFID_Cards
Fare Container
Fare Deduction
Cloud Service
Cache
Fare Deduction
Table 1:
Fare_Deductions
Table 2: Fares
Table 3: Fare
Capping
API
Gateway
Payments Container
Mock Payment
Cloud Service
Payments
Table 3:
Payments
Message Queue
Journey Container
Journey Audit
Service
Journeys
Table 1:
Journeys
Transactions Container
Transaction Record
Service
Transactions
Table 1:
Transactions