Problem Statement

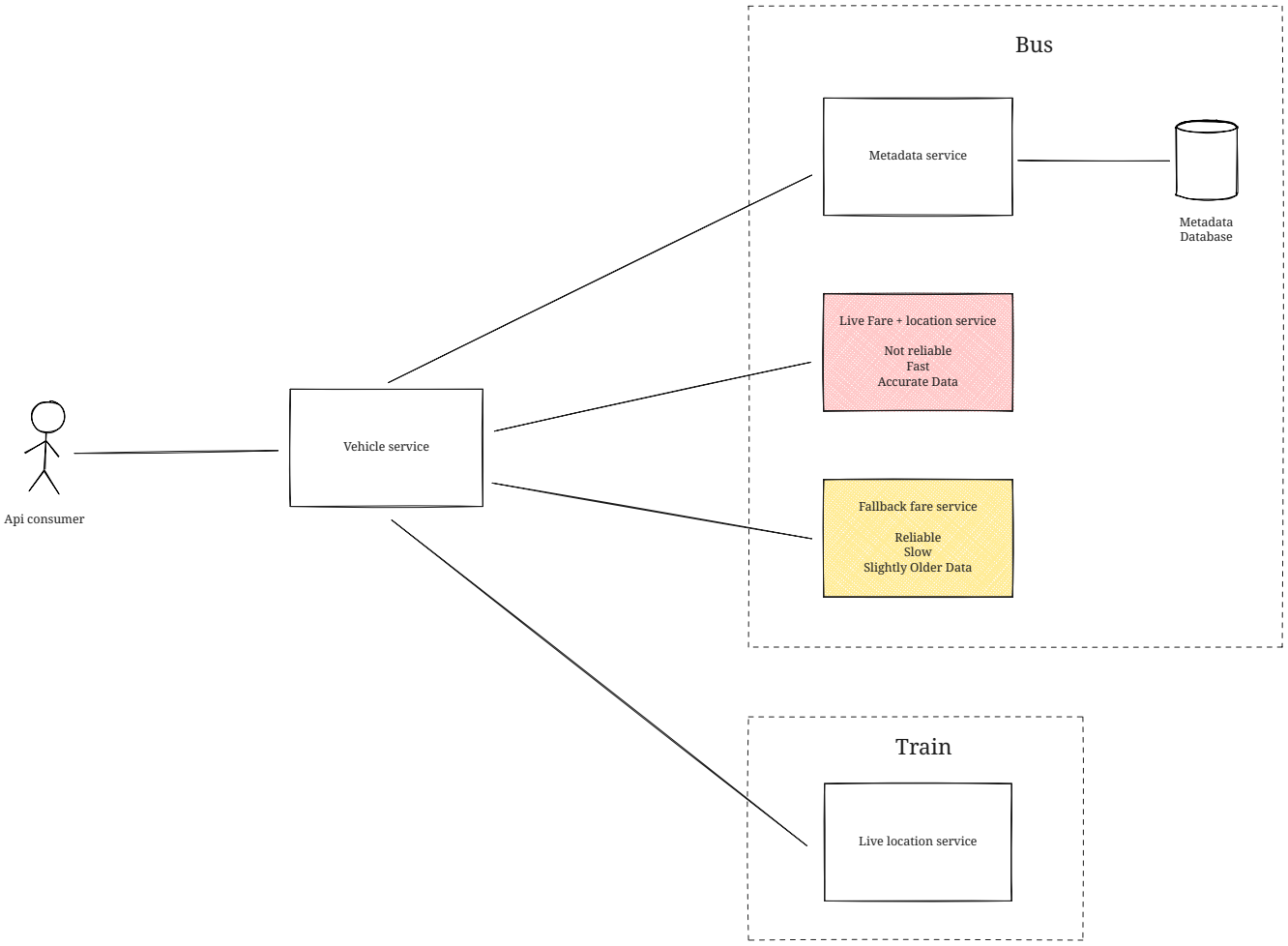
We have a city that runs its own buses & trains. They exposed their data via various microservices & some of these services are error-prone while some are reliable.

We need to build a reliable reactive aggregator service that aggregates all the data from dependent services (some of these are reactive & some are not. Refer to respective service & endpoints below for more info).

To achieve this, we need to

- Implement all end points of following services
  - `Vehicle Aggregator Service`
  - Dependent 3rd party service (we need to build appropriate mock data such that all scenarios in `Vehicle Aggregator Service` can be tested)
    - `Bus Metadata Service`
    - `Bus Live Fare & Location Service`
    - `Bus Fallback Fare Service`
    - `Train Live Location Service`
- Write integration tests for all endpoints in `Vehicle Aggregator Service`.
- Follow best practices at all levels: Java, Spring (both in Reactive & non-Reactive world), Low level design, API error handling, Build & others.

This is the top-level view of the system that needs to be built.



Here are the responsibilities of each service

Service	Responsibility
`Vehicle Aggregator Service`	Generates & aggregates various data streams, handles & auto-recovers from upstream errors.
`Bus Metadata Service`	Stores Metadata related to buses.
`Bus Live Fare & Location Service`	Provides fast & accurate realtime fare of each bus & also provides a stream of all buses. But all endpoints in this service are not reliable i.e endpoints fail often.
`Bus Fallback Fare Service`	Provides outdated fare details for each bus. Reliable (doesnt fail often), Slow & Outdated older data.
`Train Live Location Service`	Provides realtime stream of all trains. Reliable, Fast & Accurate.

Here is the outline of endpoints we need to implement (implementation constraints for each of the endpoint are mentioned in respective service section).

Service	Endpoint	What its expected to
`Vehicle Aggregator Service`	`GET /buses`	Returns list of buses along with their fares, where fares are fetched from fallback service if fetching of this info from live service fails.
	`GET /vehicle-location-stream`	Exposes an infinite reliable stream that aggregates (1) Error-prone bus location stream (auto recovers i.e resubscribes incase of upstream errors out) with (2) Reliable train location stream.
`Bus Metadata Service`	`GET /metadata`	Fetches already stored metadata about all buses.
`Bus Live Fare & Location Service`	`GET /buses/{id}/fare`	Given a bus id, we need to fetch latest fare for the vehicle in question. Fails 50% of Time.
	`GET /location-stream`	Exposes a location stream that contains events representing movement of all buses in the fleet. Each event will have location information along with bus id. Unreliable stream that emits 1 event/second & stream fails after emitting 5 items.
`Bus Fallback Fare Service`	`GET /buses/{id}/fare`	Given a bus id, we need to fetch outdated older version fare for the vehicle in question.
`Train Live Location service`	`GET /location-stream`	Exposes a location stream that contains events representing movement of all trains. Each event will have location information along with train id. Reliable infinite stream which emits 1 event/second that never fails.

Exact details of all endpoints & flows that needs to be implemented are mentioned below.

Vehicle Aggregator Service

This service is responsible for exposing API endpoints that aggregate data from dependent micro services & meet business constraints (refer to diagram above on how services are wired).

Implementation constraint

- This service has to be an end-to-end non-blocking reactive **Spring Webflux** application that aggregates data from dependent services.
- We do not need to use any database for this.

Endpoints

The following endpoint/s needs to be implemented.

#1: List of aggregated buses with fares

NOTE	Endpoint needs to be end-to-end reactive.

When API consumer makes a request to this endpoint we need to

1. Fetch list of buses from `Bus Metadata Service` using `GET /metadata` endpoint.
2. For all buses (emphasis on reducing latency), enrich this data with fare information retrieved in the following manner
   a. Attempt to fetch latest fare for bus from `Bus Live Fare & Location Service` using `GET /buses/{id}/fare` endpoint. If this fetch is successful, use it.
   b. If it fails, fetch an older outdated fare deatils from `Bus Fallback Fare Service` using `GET /buses/{id}/fare` endpoint,

Request

```
GET /buses
Accept: application/json
```

Response

```
[
  {
    "id": 1,
    "licensePlate": "BUS1",
    "fare": 101
  },
  {
    "id": 2,
    "licensePlate": "BUS2",
    "fare": 102
  },
  ...
  {
    "id": 9,
```

```
    "licensePlate": "BUS9",
    "fare": 109
  }
]
```

#2: Aggregated location stream of all vehicles (buses & trains)

NOTE	Endpoint needs to be end-to-end reactive.

When API consumer makes a request to this endpoint we need to

1. Aggregate location streams & expose a unified location stream. This endpoint aggregates
   a. Bus location stream `GET /location-stream` of `Bus Live Fare & Location Service`.
   b. Train location stream `GET /location-stream` of `Train Live Location Service`.
2. Each stream element received from the upstream is enriched with this property `type`. If the stream from which location element was received is `Bus Live Fare & Location Service`, we would mark location event's `type` as `BUS`. Else, `type` value would be `TRAIN`.

Since `GET /location-stream` of `Bus Live Fare & Location Service` is highly flaky i.e emits certain number of locations & errors out, our aggregated stream needs to auto-recover, resubscribe to same source & emit i.e our aggregated stream should be tolerant of upstream failures auto recover so our consumer never notices that the upstream emitted failures & we auto-recovered.

Request

```
GET /vehicle-location-stream
Accept: text/event-stream
```

Response

This is an infinite stream where each element in the stream will be in following formats depending on if the event emitted is from bus stream or train stream.

Event enriched from bus stream

```
{
  "id": 1,
  "type": "BUS",
  "location": {
    "lat": 11,
    "lng": 11
  }
}
```

Event enriched from train stream

```
{
  "id": 1,
  "type": "TRAIN",
  "location": {
    "lat": 91,
    "lng": 91
  }
}
```

Bus Metadata Service

This service is responsible for providing metadata about buses. Bus metadata consists of `id` & `licensePlate`.

Implementation constraint

- Implement this service as a non-Reactive Spring Web MVC application with data retrieved from any RDBMS.
- Load sample data into database on start & use Spring Data JPA when interacting with database.

Endpoints

The following endpoint/s needs to be implemented.

#1: Get Metadata

When API consumer makes a request to this endpoint we need to

- Return all bus metadata retrieved from database.

Request

```
GET /metadata
Accept: application/json
```

Response

```
[
  {
    "id": 1,
    "licensePlate": "BUS1"
  },
  {
```

```
    "id": 2,
    "licensePlate": "BUS2"
  },
  ...
  {
    "id": 9,
    "licensePlate": "BUS9"
  }
]
```

Bus Live Fare & Location Service

This service is responsible for providing these details

- Latest fare information.
- Realtime bus location stream that representing movements of all buses in the fleet.

This service is highly unreliable & all endpoints fail unexpectedly.

Implementation constraint

- This service has to be an end-to-end non-blocking reactive **Spring Webflux** application that generates mock data to test all scenarios at `Vehicle Aggregator Service` level.
- We do not need to use any database for this service.

Endpoints

The following endpoints need to be implemented.

#1: Fetch bus fare by vehicle id

NOTE	Endpoint needs to be end-to-end reactive.

When API consumer makes a request to this endpoint we need to

- Send a mock fare 50% of time.
- Error out remaining 50% of time to simulate the fact that this endpoint is unreliable.

Request

```
GET /buses/{id}/fare
Accept: application/json
```

Response

```
{
  "id": 1,
  "fare": 101
}
```

#2: Location stream

NOTE	Endpoint needs to be end-to-end reactive.

When API consumer makes a request to this endpoint we need to

- Emit 1 location per vehicle using in memory implementation & after emitting 5 items, stream errors out to simulate the fact that stream emitted by this endpoint is unreliable.

Request

```
GET /location-stream
Accept: text/event-stream
```

Response

Each element in the stream will be in the following format.

```
{
  "id": 1,
  "location": {
    "lat": 11,
    "lng": 11
  }
}
```

Bus Fallback Fare Service

This service is responsible for providing these details

- Outdated fare information.

Implementation constraint

- Implement this service as a non-Reactive Spring Web MVC application that returns mock fare info.
- We do not need to use any database for this.

Endpoints

The following endpoints need to be implemented.

#1: Fetch bus fare by vehicle id

Given a bus id we need to fetch older version of fare.

Request

```
GET /buses/{id}/fare
Accept: application/json
```

Response

```
{
  "id": 1,
  "fare": 201
}
```

Train Live Location service

This service is responsible for providing these details

- Realtime stream of locations all trains.

Implementation constraint

- This service has to be an end-to-end non-blocking reactive **Spring Webflux** application.
- We do not need to use any database for this service.

Endpoints

The following endpoints need to be implemented.

#1: Location stream

NOTE	Endpoint needs to be end-to-end reactive.

When API consumer makes a request to this endpoint we need to

- Emit 1 location per train using in memory implementation. This is an infinite stream that never fails.

Request

```
GET /location-stream
Accept: text/event-stream
```

Response

Each element in the stream will be in the following format.

```
{
  "id": 1,
  "location": {
    "lat": 91,
    "lng": 91
  }
}
```