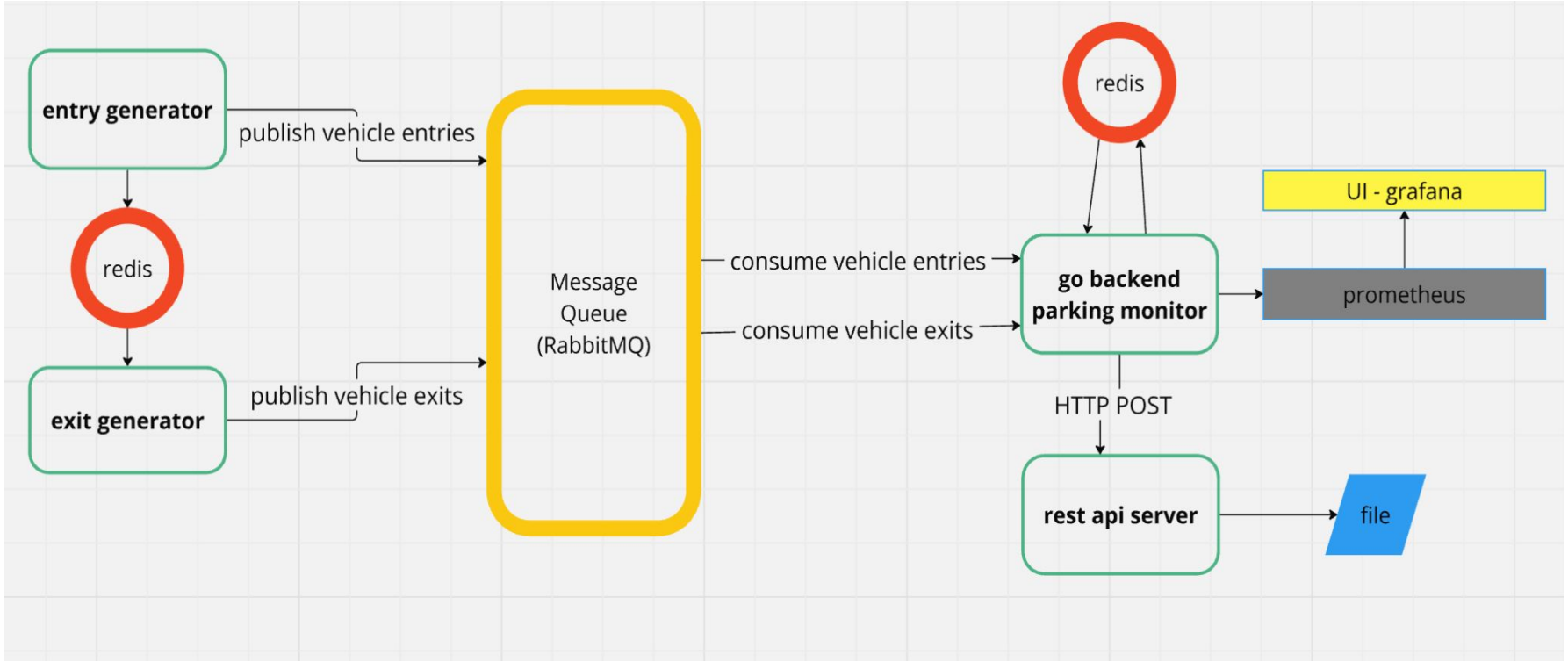
System Diagram



Implementation

Generator logic:

Entry generator -> push random generated vehicle plate to DB, messagequeue

Exit generator -> generate random number in [1,100]

If (random number > 80)

Push exit event for a plate available in DB to messagequeue

Else

Push exit event for random generated vehicle plate

Backend Logic

Follows a pipeline design pattern with dependency injection to facilitate testing as well as decoupling from dependant libraries (rabbitmq, redisdb, restapi client)

Consumer from rabbitmq creates a go channel which gets populated with messages as they arrive.

Consumer func for a queue, in a goroutine, loops through the messages and calls the handler function for the message.

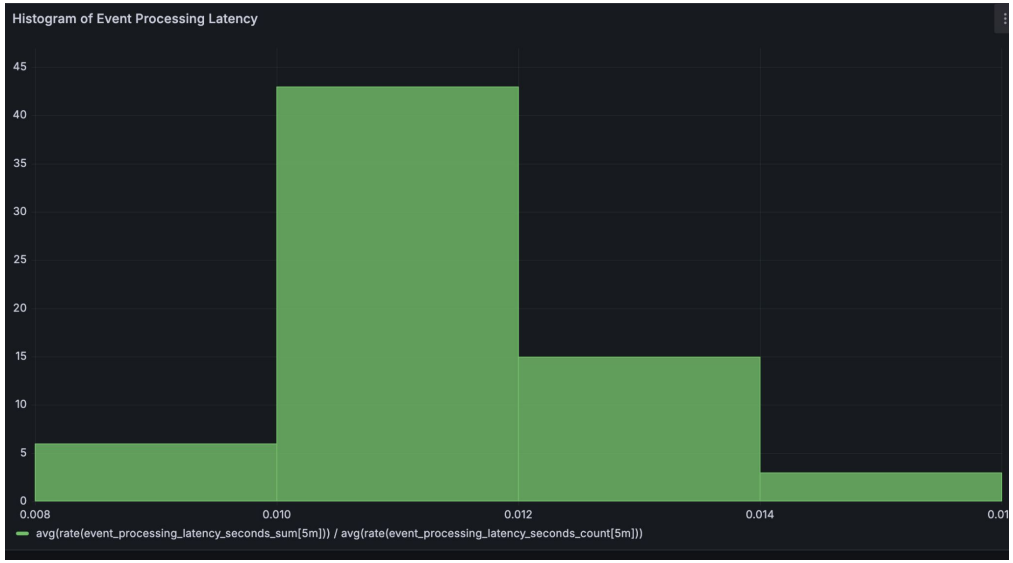
Handler func is declared in an interface

So EntryEventProcessor and ExitEventProcessor are created to implement the Handler interface . and because of goroutines are handled concurrently

```
Consume message( messagehandler)
        Loop:msgs
                messagehandler.ProcessMessage(msg)
messagehandler.ProcessMessage(msg)
        msg.parse
        dataStore. Store/Retrieve
        (Calculate summary)
        (Restapi.post summary)
```

Metrics

MessageHandler.ProcessMessage is instrumented to get event processing latency metrics



Average event processing time over a 5m interval is calculated and plotted as histogram.

Majority have the latency of 0.01 to 0.02 seconds