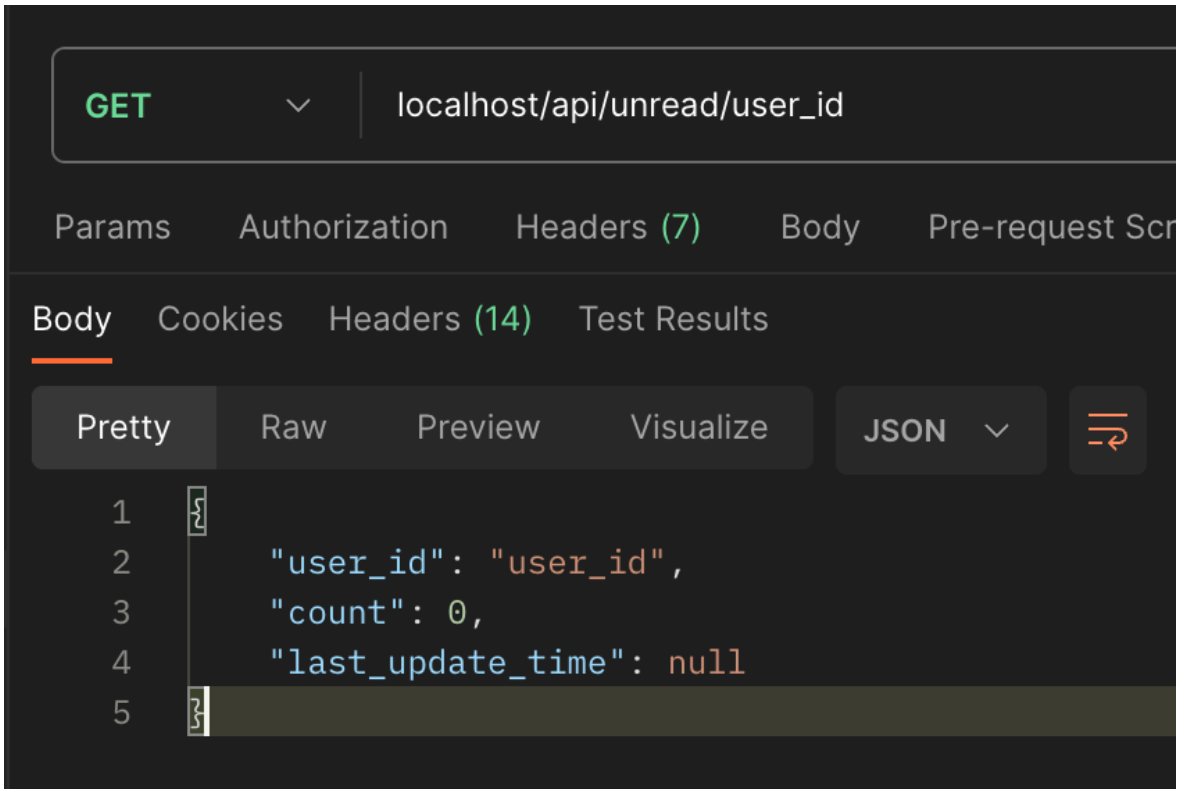
Saga-паттерн

Согласно заданию, выделим модуль счетчика, который будет хранить в плоской таблице идентификатор пользователя и количество непрочитанных сообщений.
Исходя из поставлено задачи нужно решить две проблемы: обеспечить высокую доступность данных счетчиков на чтение и консистентность данных между счетчиком и количеством непрочитанных сообщений. Разберем решение по шагам.

Обеспечение высокой доступности данных
1. Для отображения данных в модуле счетчика создадим rest-эндпойнт для получения текущего количества непрочитанных данных по user_id, который будет возвращать в ответ json:



Для упрощения задачи сделаем общедоступным этот эндпойнт, оставим за пределами скоупа задачи авторизацию и ролевую модель.
2. Внутри модуля создадим кеш. При первом обращении будет осуществляться проверка записи в кеше, если записи нет - получим количество данных из БД и положим в кеш.

```
@CachePut(cacheNames = UNREAD_MSG_COUNTER_CACHE, key = "#dto.userId")
public CounterDto updateUnreadMsgCounter(CounterDto dto) {...}

@CachePut(cacheNames = UNREAD_MSG_COUNTER_CACHE, key = "#userId")
public CounterDto getUnreadMsgCounter(String userId) {
    var counterDto = cacheService.getFromCache(UNREAD_MSG_COUNTER_CACHE, userId, CounterDto.class);
    if (counterDto == null) {
        return Optional.of(getAllCountsFromDatabase(userId)) Optional<Integer>
                .map(count -> new CounterDto()
                        .setCount(count)
                        .setUserId(userId)) Optional<CounterDto>
                .get();
    }
    return counterDto;
}
```

Таким образом, чтение будет происходить из кеша при наличии записи, что снизит нагрузку на БД и сократит время ответа. При желании - функционал кеша можно вынести в отдельной стоящий сервис типа Redis, но в данном решении для упрощения используется in-memory кеширование стандартными средствами.

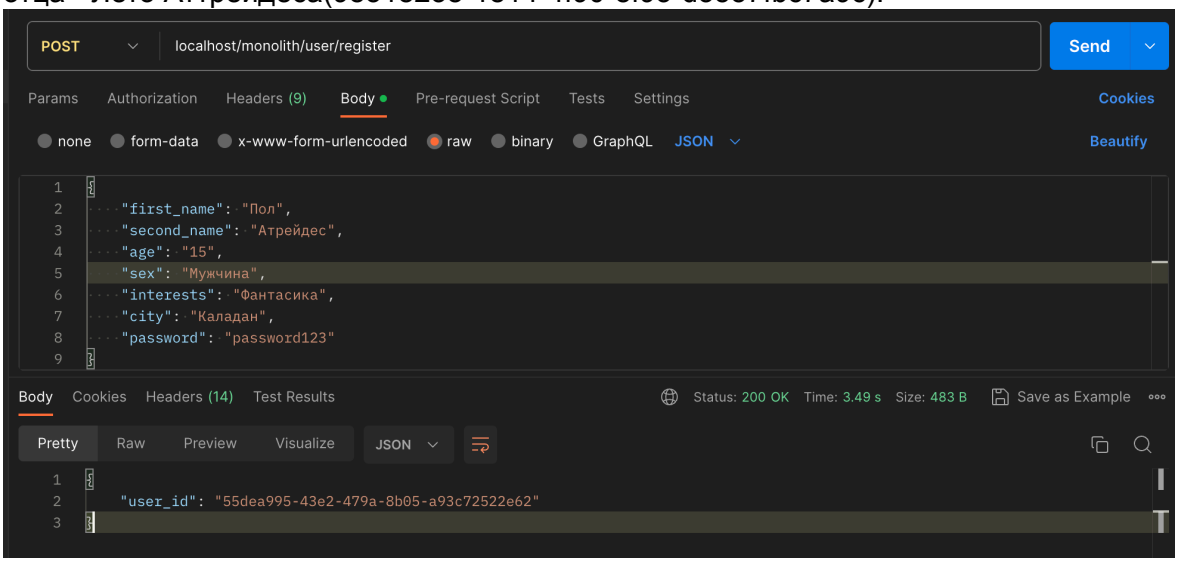
Обеспечение консистентности данных

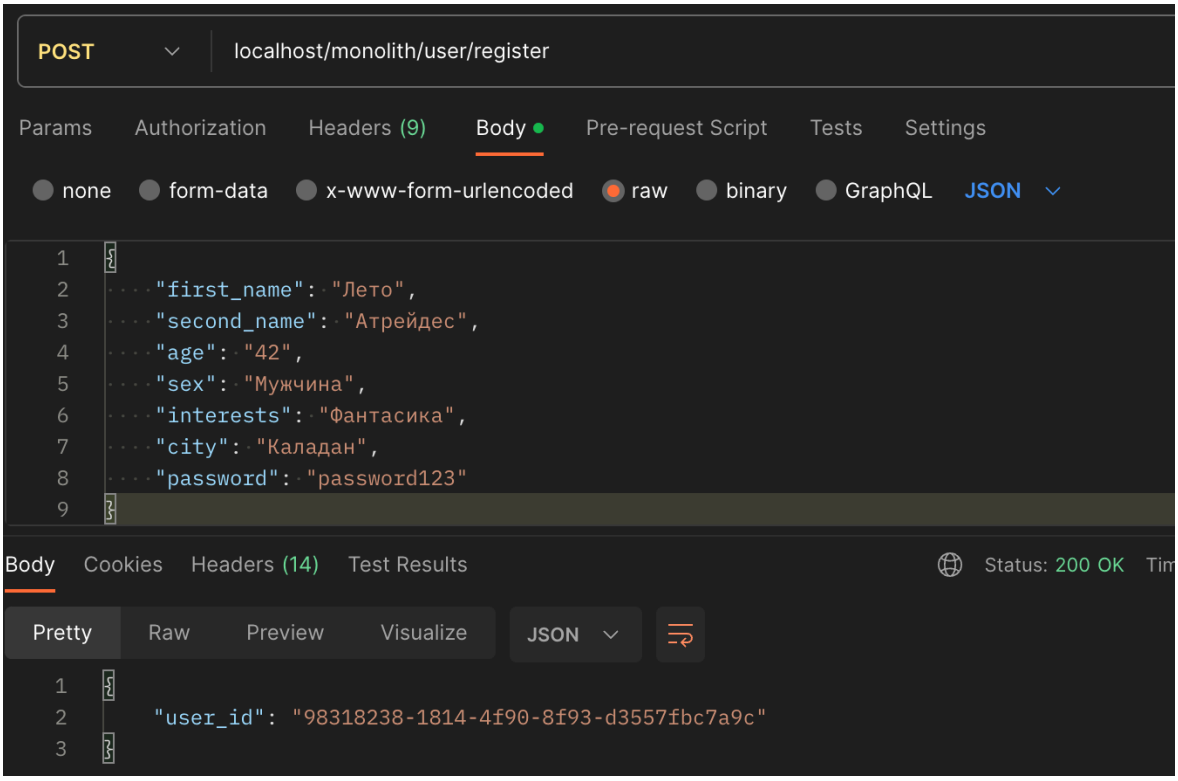
Основная проблема здесь в том, что событие записи нового сообщение и обновление счетчика могут происходить либо синхронно, либо нет. Первый путь предполагает, что все изменения происходят в одной транзакции, но при этом между двумя микросервисами. В текущем решении инстанс БД один для простоты решения, но предполагается что кажды сервис имеет свой независимый инстанс. В таком случае либо два микросервиса нужно объединить в монолит, либо использовать распределенную транзакцию со всеми вытекающими минусами.

Для решения используем паттерн Saga с хореографией.

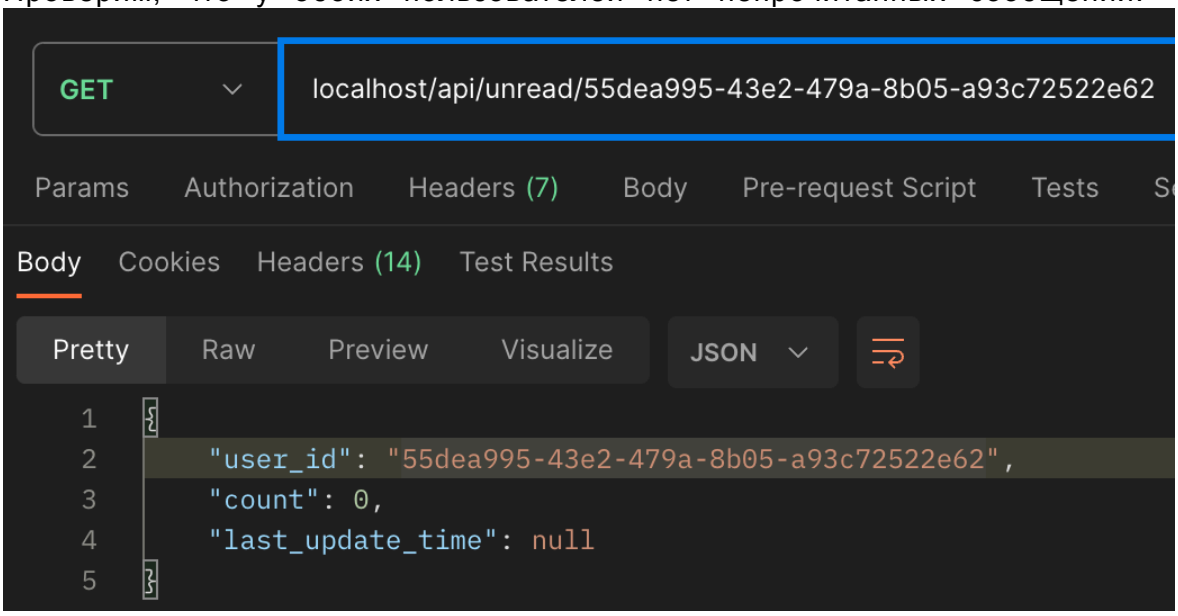
Рассмотрим подробнее на примере, выполним следующие шаги:

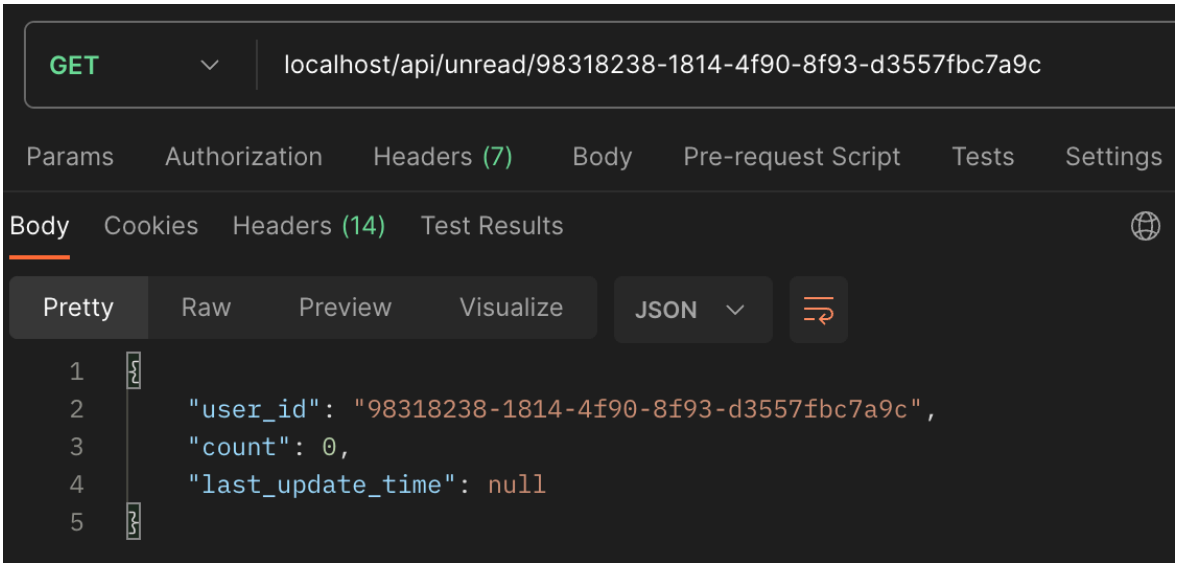
1. Запустим необходимое окружение согласно readme.md проекта.
2. Используя коллекцию Postman из корня проекта зарегистрируем двух пользователей - Пола Аттрейдеса (55dea995-43e2-479a-8b05-a93c72522e62) и его отца - Лето Аттрейдеса(98318238-1814-4f90-8f93-d3557fbc7a9c).



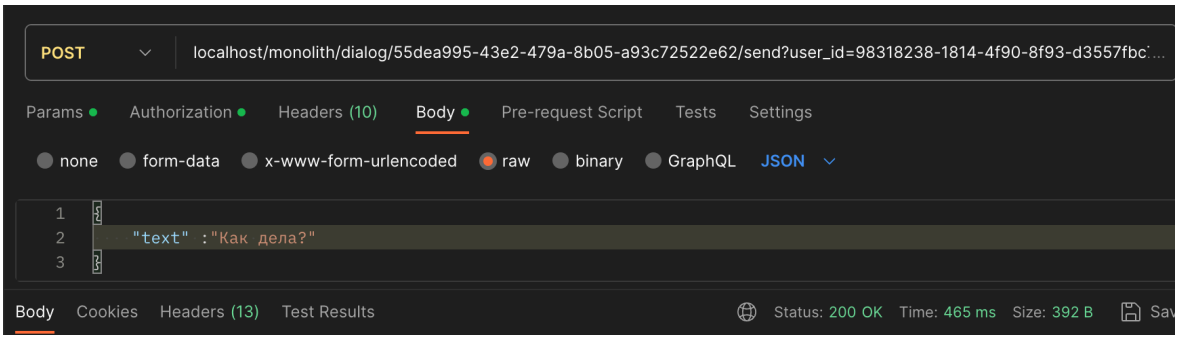


3. Проверим, что у обоих пользователей нет непрочитанных сообщений.

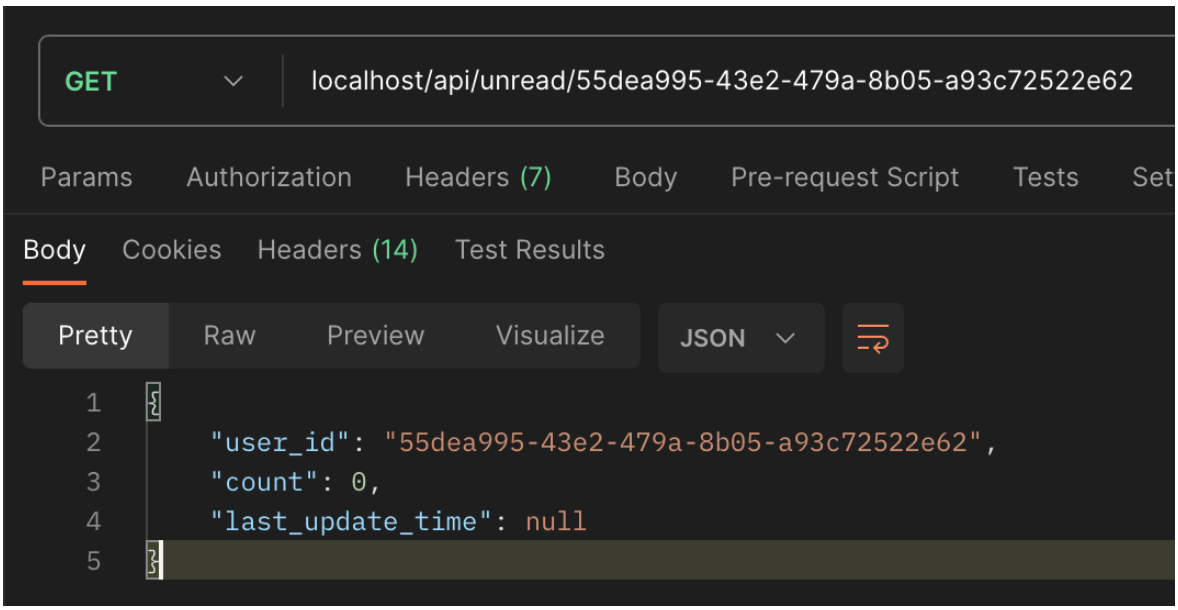


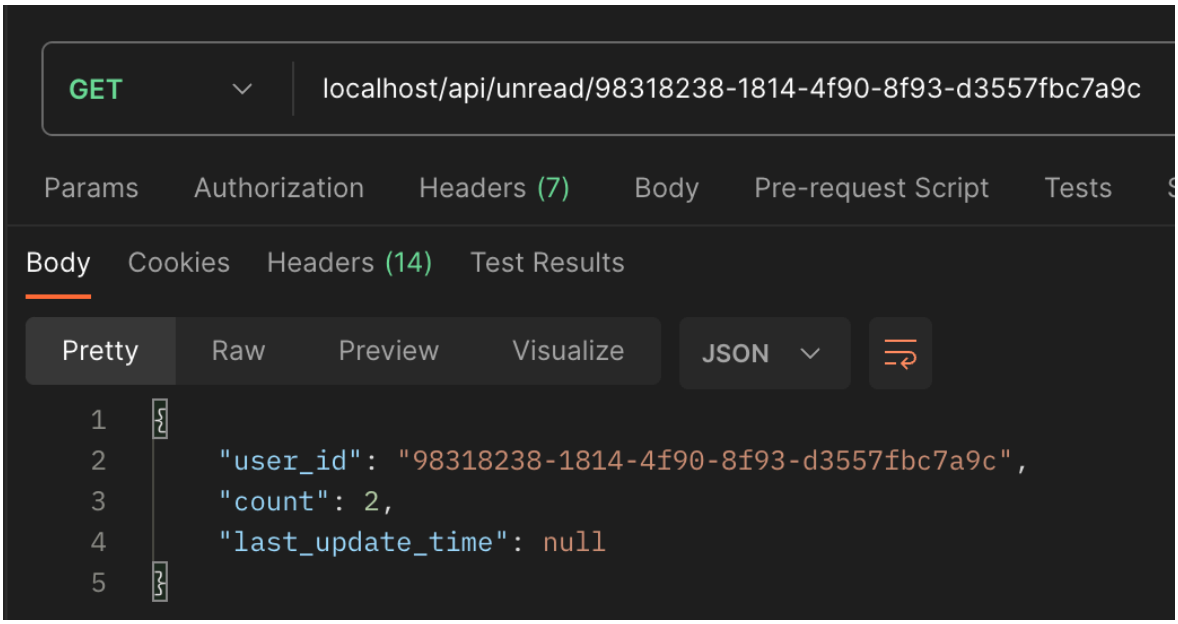


4. Начнем диалог от имени Пола аттрейдеса, отправив пару сообщений («Привет!», «Как дела?»), выполнив перед этим вход в систему:



5. Снова запросим количество непрочитанных сообщений, увидим что у Пола счетчик по-прежнему 0, а вот у его отца - Лето - 2. Что верно, так как сообщения адресованные Лето не прочитаны, а Пол - их автор.



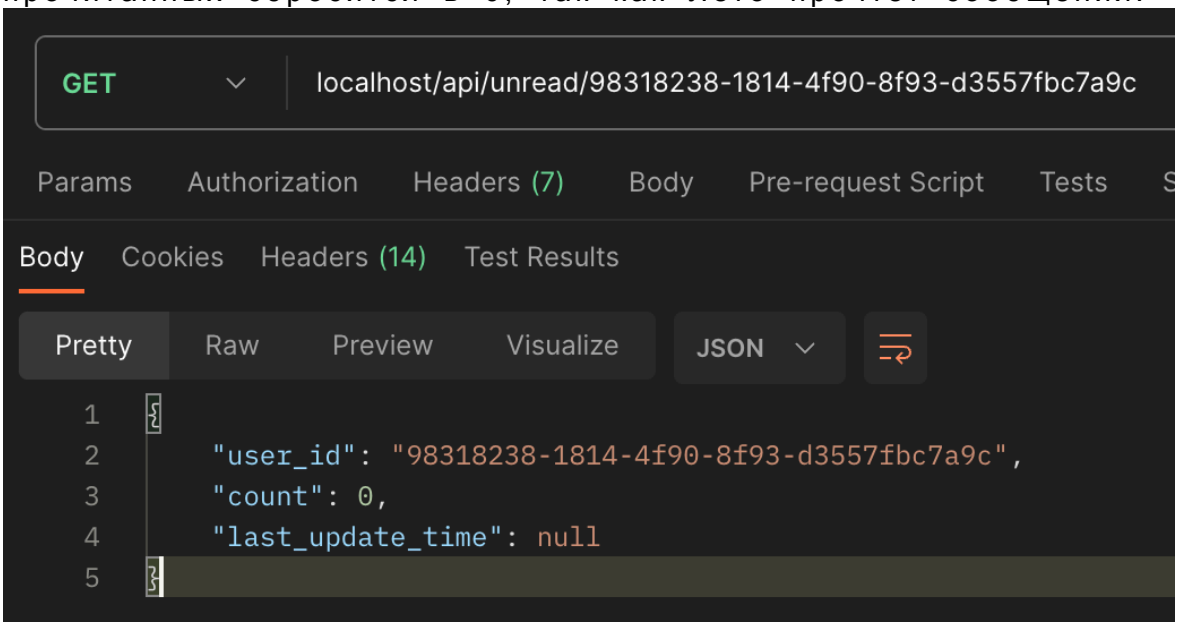


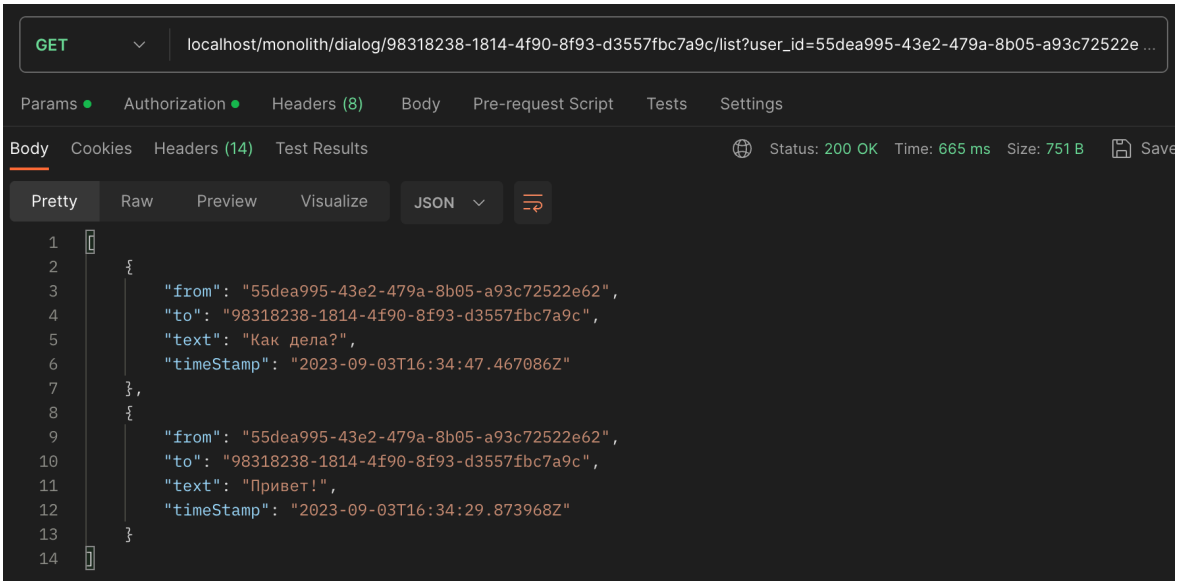
6. Проверим, что эти данные счетчика берутся из кеша, для этого обратимся к логу приложения:

2023-09-03 19:52:46 2023-09-03 16:52:46.288 WARN 1 --- [nio-8080-exec-1] o.o.e.s.n.c.s.UnreadMessageCountService : Для пользователя 55dea995-43e2-479a-8b05-a93c72522e62 в кеше не найден счетчик, обращаемся к БД
2023-09-03 19:52:46 2023-09-03 16:52:46.303 INFO 1 --- [nio-8080-exec-1] **o.o.e.s.n.c.s.UnreadMessageCountService : Для пользователя 55dea995-43e2-479a-8b05-a93c72522e62 запрашиваем данные из БД**
2023-09-03 19:52:46 2023-09-03 16:52:46.344 INFO 1 --- [nio-8080-exec-1] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2023-09-03 19:52:46 2023-09-03 16:52:46.519 INFO 1 --- [nio-8080-exec-1] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2023-09-03 19:52:52 2023-09-03 16:52:52.512 WARN 1 --- [nio-8080-exec-2] o.o.e.s.n.c.s.UnreadMessageCountService : Для пользователя 98318238-1814-4f90-8f93-d3557fbc7a9c в кеше не найден счетчик, обращаемся к БД
2023-09-03 19:52:52 2023-09-03 16:52:52.514 INFO 1 --- [nio-8080-exec-2] o.o.e.s.n.c.s.UnreadMessageCountService : Для пользователя 98318238-1814-4f90-8f93-d3557fbc7a9c запрашиваем данные из БД

Заметно, что поскольку этих данных не было в кеше, то был запрос к БД. Этой записи в логах не будет, как и обращения к базе данных.

7. Теперь запросим диалог от имени Лето Аттрейдеса и убедимся, что счетчик прочитанных сбросится в 0, так как Лето прочтет сообщения:





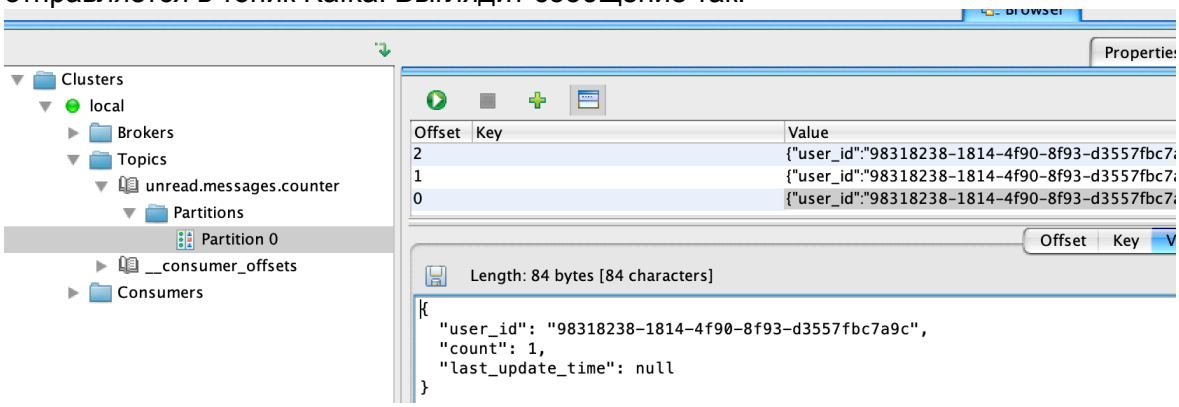
Видим, что Лето успешно получил диалог, увидел два сообщения и счетчик сбросился в 0. Причем обращение за счетчиком непрочитанных для Лето заканчивается на запросе данных из кеша, судя по логам и отсутствующему соответствующему сообщению:

2023-09-03 20:00:23 2023-09-03 17:00:23.554 WARN 1 --- [ntainer#0-0-C-1] com.zaxxer.hikari.pool.PoolBase : HikariPool-1 - Failed to validate connection org.postgresql.jdbc.PgConnection@47d8ede5 (This connection has been closed.). Possibly consider using a shorter maxLifetime value.
2023-09-03 20:00:23 2023-09-03 17:00:23.554 WARN 1 --- [ntainer#0-0-C-1] com.zaxxer.hikari.pool.PoolBase : HikariPool-1 - Failed to validate connection org.postgresql.jdbc.PgConnection@7a64fd79 (This connection has been closed.). Possibly consider using a shorter maxLifetime value.
2023-09-03 20:00:23 2023-09-03 17:00:23.652 INFO 1 --- [ntainer#0-0-C-1] o.o.e.s.n.c.s.UnreadMessageCountService : Измененное количество: CounterDto(userId=98318238-1814-4f90-8f93-d3557fbc7a9c, count=0, lastUpdateTime=null)
2023-09-03 20:01:02 2023-09-03 17:01:02.983 WARN 1 --- [nio-8080-exec-2] o.o.e.s.n.c.s.UnreadMessageCountService : Для пользователя 55dea995-43e2-479a-8b05-a93c72522e62 в кеше не найден счетчик, обращаемся к БД
2023-09-03 20:01:02 2023-09-03 17:01:02.986 INFO 1 --- [nio-8080-exec-2] o.o.e.s.n.c.s.UnreadMessageCountService : Для пользователя 55dea995-43e2-479a-8b05-a93c72522e62 запрашиваем данные из БД

Видим только обращение к БД за счетчиком Пола, поскольку ТТЛ записи истек пока мы выполняли манипуляции с чтением.

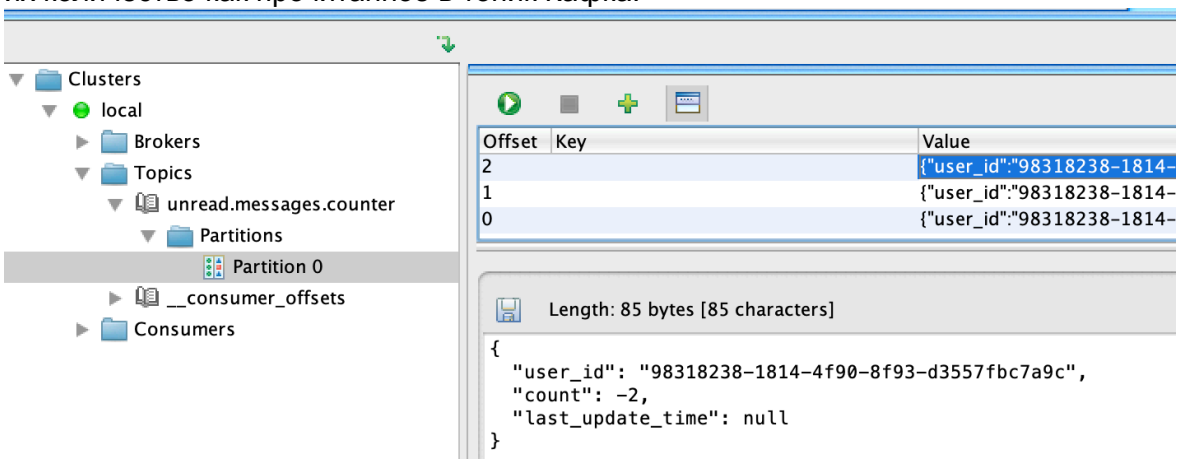
Таким образом, данные остаются консистентными. Для этого реализован механизм хореографии Saga. Разберем по шагам, события:

1. В момент, когда создается сообщение - внутри сервиса диалогов формируется событие для сервиса счетчиков, об инкремент непрочитанных сообщений и отправляется в топик Kafka. Выглядит сообщение так:



2. Сервис счетчиков получает это сообщение, мерджит его с тем, что хранится в БД (если данных в таблице нет, считаем что текущее значение - 0). И сохраняет новое значение в базе и кеше.

3. При запросе переписки сервис диалогов извлекает из БД сообщения и оправляет их количество как прочитанное в топик Кафка:



4. Сервис счетчиков получает сообщение, снова мерджит данные и кладет в базу и

кеш обновленное значение:

```
o.o.e.s.n.counter.service.KafkaConsumer  : Сообщение из топика:
{"user_id":"98318238-1814-4f90-8f93-d3557fbc7a9c","count":-2,"last_update_time":null}
2023-09-03 20:00:23 2023-09-03 17:00:23.523  INFO 1 --- [ntainer#0-0-C-1]
o.o.e.s.n.counter.service.KafkaConsumer  : Смапленное сообщение:
CounterDto(userId=98318238-1814-4f90-8f93-d3557fbc7a9c, count=-2, lastUpdateTime=null)
2023-09-03 20:00:23 2023-09-03 17:00:23.652  INFO 1 --- [ntainer#0-0-C-1]
o.o.e.s.n.c.s.UnreadMessageCountService  : Измененное количество:
CounterDto(userId=98318238-1814-4f90-8f93-d3557fbc7a9c, count=0, lastUpdateTime=null)
```

Выводы:

1. Сервис диалогов генерирует события для сервиса счетчиков, тот в свою очередь - генерирует обновления для эндпойнта счетчиков. Это и есть оркестрация Saga с определенными ограничениями.
2. В текущей реализации не предусмотрены сторнирующие транзакции: между отправкой события от сервиса и фактической передачей данных от монолита дальше есть вероятность отказа монолита. То есть данные могли быть вычитаны из БД, но быть потеряны по-пути и не отправлены пользователю. Здесь должна помочь сторнирующая транзакция на случай ошибки. Но виду бедности статусной модели сообщения (отсутствие статуса прочитан/непрочитан, механизма акноледжа события прочитан/не прочитан и так далее) эту задачу оставим за скобками.
3. Для упрощения примера сервис отображения счетчика и кеш, а также сами счетчики реализованы в рамках одного модуля, но при желании их можно разделить, увеличив тем самым событийную цепочку Saga - при обновлении данных в базе, затем сгенерировать новое сообщение Kafka, на которое отреагирует сервис отображения счетчика - получит новой сообщение и обновит кеш. Если кеш распределенный - то сервис счетчиков может без генерации сообщения напрямую обновить кеш, а сервис отображения - просто возьмет обновленное значение при следующем запросе.