package com.twitter.tweetypie

package service

package observer

import com.twitter.tweetypie.thriftscala.GetStoredTweetsByUserRequest

import com.twitter.tweetypie.thriftscala.GetStoredTweetsByUserResult

private[service] object GetStoredTweetsByUserObserver extends StoredTweetsObserver {

type Type = ObserveExchange[GetStoredTweetsByUserRequest, GetStoredTweetsByUserResult]

val firstTweetTimestamp: Long = 1142974200L

def observeRequest(stats: StatsReceiver): Effect[GetStoredTweetsByUserRequest] = {

val optionsScope = stats.scope("options")

val bypassVisibilityFilteringCounter = optionsScope.counter("bypass\_visibility\_filtering")

val forUserIdCounter = optionsScope.counter("set\_for\_user\_id")

val timeRangeStat = optionsScope.stat("time\_range\_seconds")

val cursorCounter = optionsScope.counter("cursor")

val startFromOldestCounter = optionsScope.counter("start\_from\_oldest")

val additionalFieldsScope = optionsScope.scope("additional\_fields")

Effect { request =>

if (request.options.isDefined) {

val options = request.options.get

if (options.bypassVisibilityFiltering) bypassVisibilityFilteringCounter.incr()

if (options.setForUserId) forUserIdCounter.incr()

if (options.cursor.isDefined) {

cursorCounter.incr()

} else {

// We only add a time range stat once, when there's no cursor in the request (i.e. this

// isn't a repeat request for a subsequent batch of results)

val startTimeSeconds: Long =

options.startTimeMsec.map(\_ / 1000).getOrElse(firstTweetTimestamp)

val endTimeSeconds: Long = options.endTimeMsec.map(\_ / 1000).getOrElse(Time.now.inSeconds)

timeRangeStat.add(endTimeSeconds - startTimeSeconds)

// We use the startFromOldest parameter when the cursor isn't defined

if (options.startFromOldest) startFromOldestCounter.incr()

}

options.additionalFieldIds.foreach { id =>

additionalFieldsScope.counter(id.toString).incr()

}

}

}

}

def observeResult(stats: StatsReceiver): Effect[GetStoredTweetsByUserResult] = {

val resultScope = stats.scope("result")

Effect { result =>

observeStoredTweets(result.storedTweets, resultScope)

}

}

def observeExchange(stats: StatsReceiver): Effect[Type] = {

val resultStateStats = ResultStateStats(stats)

Effect {

case (request, response) =>

response match {

case Return(\_) => resultStateStats.success()

case Throw(\_) => resultStateStats.failed()

}

}

}

}