package com.twitter.frigate.pushservice.store

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.onboarding.task.service.thriftscala.FatigueFlowEnrollment

import com.twitter.stitch.Stitch

import com.twitter.storage.client.manhattan.bijections.Bijections.BinaryScalaInjection

import com.twitter.storage.client.manhattan.bijections.Bijections.LongInjection

import com.twitter.storage.client.manhattan.bijections.Bijections.StringInjection

import com.twitter.storage.client.manhattan.kv.impl.Component

import com.twitter.storage.client.manhattan.kv.impl.KeyDescriptor

import com.twitter.storage.client.manhattan.kv.impl.ValueDescriptor

import com.twitter.storage.client.manhattan.kv.ManhattanKVClient

import com.twitter.storage.client.manhattan.kv.ManhattanKVClientMtlsParams

import com.twitter.storage.client.manhattan.kv.ManhattanKVEndpointBuilder

import com.twitter.storage.client.manhattan.kv.NoMtlsParams

import com.twitter.storehaus.ReadableStore

import com.twitter.storehaus\_internal.manhattan.Omega

import com.twitter.util.Duration

import com.twitter.util.Future

import com.twitter.util.Time

case class OCFHistoryStoreKey(userId: Long, fatigueDuration: Duration, fatigueGroup: String)

class OCFPromptHistoryStore(

manhattanAppId: String,

dataset: String,

mtlsParams: ManhattanKVClientMtlsParams = NoMtlsParams

)(

implicit stats: StatsReceiver)

extends ReadableStore[OCFHistoryStoreKey, FatigueFlowEnrollment] {

import ManhattanInjections.\_

private val client = ManhattanKVClient(

appId = manhattanAppId,

dest = Omega.wilyName,

mtlsParams = mtlsParams,

label = "ocf\_history\_store"

)

private val endpoint = ManhattanKVEndpointBuilder(client, defaultMaxTimeout = 5.seconds)

.statsReceiver(stats.scope("ocf\_history\_store"))

.build()

private val limitResultsTo = 1

private val datasetKey = keyDesc.withDataset(dataset)

override def get(storeKey: OCFHistoryStoreKey): Future[Option[FatigueFlowEnrollment]] = {

val userId = storeKey.userId

val fatigueGroup = storeKey.fatigueGroup

val fatigueLength = storeKey.fatigueDuration.inMilliseconds

val currentTime = Time.now.inMilliseconds

val fullKey = datasetKey

.withPkey(userId)

.from(fatigueGroup)

.to(fatigueGroup, fatigueLength - currentTime)

Stitch

.run(endpoint.slice(fullKey, valDesc, limit = Some(limitResultsTo)))

.map { results =>

if (results.nonEmpty) {

val (\_, mhValue) = results.head

Some(mhValue.contents)

} else None

}

}

}

object ManhattanInjections {

val keyDesc = KeyDescriptor(Component(LongInjection), Component(StringInjection, LongInjection))

val valDesc = ValueDescriptor(BinaryScalaInjection(FatigueFlowEnrollment))

}