package com.twitter.tsp.stores

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.store.InterestedInInterestsFetchKey

import com.twitter.frigate.common.store.strato.StratoFetchableStore

import com.twitter.hermit.store.common.ObservedReadableStore

import com.twitter.interests.thriftscala.InterestId

import com.twitter.interests.thriftscala.InterestLabel

import com.twitter.interests.thriftscala.InterestRelationship

import com.twitter.interests.thriftscala.InterestRelationshipV1

import com.twitter.interests.thriftscala.InterestedInInterestLookupContext

import com.twitter.interests.thriftscala.InterestedInInterestModel

import com.twitter.interests.thriftscala.OptOutInterestLookupContext

import com.twitter.interests.thriftscala.UserInterest

import com.twitter.interests.thriftscala.UserInterestData

import com.twitter.interests.thriftscala.UserInterestsResponse

import com.twitter.simclusters\_v2.common.UserId

import com.twitter.storehaus.ReadableStore

import com.twitter.strato.client.Client

import com.twitter.strato.thrift.ScroogeConvImplicits.\_

case class TopicResponse(

entityId: Long,

interestedInData: Seq[InterestedInInterestModel],

scoreOverride: Option[Double] = None,

notInterestedInTimestamp: Option[Long] = None,

topicFollowTimestamp: Option[Long] = None)

case class TopicResponses(responses: Seq[TopicResponse])

object TopicStore {

private val InterestedInInterestsColumn = "interests/interestedInInterests"

private lazy val ExplicitInterestsContext: InterestedInInterestLookupContext =

InterestedInInterestLookupContext(

explicitContext = None,

inferredContext = None,

disableImplicit = Some(true)

)

private def userInterestsResponseToTopicResponse(

userInterestsResponse: UserInterestsResponse

): TopicResponses = {

val responses = userInterestsResponse.interests.interests.toSeq.flatMap { userInterests =>

userInterests.collect {

case UserInterest(

InterestId.SemanticCore(semanticCoreEntity),

Some(UserInterestData.InterestedIn(data))) =>

val topicFollowingTimestampOpt = data.collect {

case InterestedInInterestModel.ExplicitModel(

InterestRelationship.V1(interestRelationshipV1)) =>

interestRelationshipV1.timestampMs

}.lastOption

TopicResponse(semanticCoreEntity.id, data, None, None, topicFollowingTimestampOpt)

}

}

TopicResponses(responses)

}

def explicitFollowingTopicStore(

stratoClient: Client

)(

implicit statsReceiver: StatsReceiver

): ReadableStore[UserId, TopicResponses] = {

val stratoStore =

StratoFetchableStore

.withUnitView[InterestedInInterestsFetchKey, UserInterestsResponse](

stratoClient,

InterestedInInterestsColumn)

.composeKeyMapping[UserId](uid =>

InterestedInInterestsFetchKey(

userId = uid,

labels = None,

lookupContext = Some(ExplicitInterestsContext)

))

.mapValues(userInterestsResponseToTopicResponse)

ObservedReadableStore(stratoStore)

}

def userOptOutTopicStore(

stratoClient: Client,

optOutStratoStorePath: String

)(

implicit statsReceiver: StatsReceiver

): ReadableStore[UserId, TopicResponses] = {

val stratoStore =

StratoFetchableStore

.withUnitView[

(Long, Option[Seq[InterestLabel]], Option[OptOutInterestLookupContext]),

UserInterestsResponse](stratoClient, optOutStratoStorePath)

.composeKeyMapping[UserId](uid => (uid, None, None))

.mapValues { userInterestsResponse =>

val responses = userInterestsResponse.interests.interests.toSeq.flatMap { userInterests =>

userInterests.collect {

case UserInterest(

InterestId.SemanticCore(semanticCoreEntity),

Some(UserInterestData.InterestedIn(data))) =>

TopicResponse(semanticCoreEntity.id, data, None)

}

}

TopicResponses(responses)

}

ObservedReadableStore(stratoStore)

}

def notInterestedInTopicsStore(

stratoClient: Client,

notInterestedInStorePath: String

)(

implicit statsReceiver: StatsReceiver

): ReadableStore[UserId, TopicResponses] = {

val stratoStore =

StratoFetchableStore

.withUnitView[Long, Seq[UserInterest]](stratoClient, notInterestedInStorePath)

.composeKeyMapping[UserId](identity)

.mapValues { notInterestedInInterests =>

val responses = notInterestedInInterests.collect {

case UserInterest(

InterestId.SemanticCore(semanticCoreEntity),

Some(UserInterestData.NotInterested(notInterestedInData))) =>

val notInterestedInTimestampOpt = notInterestedInData.collect {

case InterestRelationship.V1(interestRelationshipV1: InterestRelationshipV1) =>

interestRelationshipV1.timestampMs

}.lastOption

TopicResponse(semanticCoreEntity.id, Seq.empty, None, notInterestedInTimestampOpt)

}

TopicResponses(responses)

}

ObservedReadableStore(stratoStore)

}

}