package com.twitter.usersignalservice.signals

import com.twitter.finagle.stats.StatsReceiver

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

import com.twitter.simclusters\_v2.thriftscala.InternalId

import com.twitter.strato.client.Client

import com.twitter.strato.data.Conv

import com.twitter.strato.thrift.ScroogeConv

import com.twitter.twistly.common.TwistlyProfile

import com.twitter.twistly.thriftscala.EngagementMetadata.FavoriteMetadata

import com.twitter.twistly.thriftscala.RecentEngagedTweet

import com.twitter.twistly.thriftscala.UserRecentEngagedTweets

import com.twitter.usersignalservice.base.Query

import com.twitter.usersignalservice.base.StratoSignalFetcher

import com.twitter.usersignalservice.thriftscala.Signal

import com.twitter.usersignalservice.thriftscala.SignalType

import com.twitter.util.Future

import com.twitter.util.Timer

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

case class TweetFavoritesFetcher @Inject() (

stratoClient: Client,

timer: Timer,

stats: StatsReceiver)

extends StratoSignalFetcher[(UserId, Long), Unit, UserRecentEngagedTweets] {

import TweetFavoritesFetcher.\_

override type RawSignalType = RecentEngagedTweet

override val name: String = this.getClass.getCanonicalName

override val statsReceiver: StatsReceiver = stats.scope(name)

override val stratoColumnPath: String =

TwistlyProfile.TwistlyProdProfile.userRecentEngagedStorePath

override val stratoView: Unit = None

override protected val keyConv: Conv[(UserId, Long)] = Conv.ofType

override protected val viewConv: Conv[Unit] = Conv.ofType

override protected val valueConv: Conv[UserRecentEngagedTweets] =

ScroogeConv.fromStruct[UserRecentEngagedTweets]

override protected def toStratoKey(userId: UserId): (UserId, Long) = (userId, DefaultVersion)

override protected def toRawSignals(

userRecentEngagedTweets: UserRecentEngagedTweets

): Seq[RawSignalType] =

userRecentEngagedTweets.recentEngagedTweets

override def process(

query: Query,

rawSignals: Future[Option[Seq[RawSignalType]]]

): Future[Option[Seq[Signal]]] = {

rawSignals.map {

\_.map { signals =>

val lookBackWindowFilteredSignals =

SignalFilter.lookBackWindow90DayFilter(signals, query.signalType)

lookBackWindowFilteredSignals

.filter { recentEngagedTweet =>

recentEngagedTweet.features.statusCounts

.flatMap(\_.favoriteCount).exists(\_ >= MinFavCount)

}.filter { recentEngagedTweet =>

applySignalTweetTypeFilter(query.signalType, recentEngagedTweet)

}.collect {

case RecentEngagedTweet(tweetId, engagedAt, \_: FavoriteMetadata, \_) =>

Signal(query.signalType, engagedAt, Some(InternalId.TweetId(tweetId)))

}.take(query.maxResults.getOrElse(Int.MaxValue))

}

}

}

private def applySignalTweetTypeFilter(

signal: SignalType,

recentEngagedTweet: RecentEngagedTweet

): Boolean = {

// Perform specific filters for particular signal types.

signal match {

case SignalType.AdFavorite => SignalFilter.isPromotedTweet(recentEngagedTweet)

case \_ => true

}

}

}

object TweetFavoritesFetcher {

private val MinFavCount = 10

// see com.twitter.twistly.store.UserRecentEngagedTweetsStore

private val DefaultVersion = 0

}