package com.twitter.home\_mixer.functional\_component.query\_transformer

import com.twitter.common\_internal.analytics.twitter\_client\_user\_agent\_parser.UserAgent

import com.twitter.conversions.DurationOps.\_

import com.twitter.home\_mixer.model.HomeFeatures.PersistenceEntriesFeature

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.functional\_component.transformer.CandidatePipelineQueryTransformer

import com.twitter.product\_mixer.core.model.common.identifier.TransformerIdentifier

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

import com.twitter.timelinemixer.clients.persistence.EntryWithItemIds

import com.twitter.timelines.persistence.thriftscala.RequestType

import com.twitter.timelines.util.client\_info.ClientPlatform

import com.twitter.timelineservice.model.rich.EntityIdType

import com.twitter.util.Time

object EditedTweetsCandidatePipelineQueryTransformer

extends CandidatePipelineQueryTransformer[PipelineQuery, Seq[Long]] {

override val identifier: TransformerIdentifier = TransformerIdentifier("EditedTweets")

// The time window for which a tweet remains editable after creation.

private val EditTimeWindow = 60.minutes

override def transform(query: PipelineQuery): Seq[Long] = {

val applicableCandidates = getApplicableCandidates(query)

if (applicableCandidates.nonEmpty) {

// Include the response corresponding with the Previous Timeline Load (PTL).

// Any tweets in it could have become stale since being served.

val previousTimelineLoadTime = applicableCandidates.head.servedTime

// The time window for editing a tweet is 60 minutes,

// so we ignore responses older than (PTL Time - 60 mins).

val inWindowCandidates: Seq[PersistenceStoreEntry] = applicableCandidates

.takeWhile(\_.servedTime.until(previousTimelineLoadTime) < EditTimeWindow)

// Exclude the tweet IDs for which ReplaceEntry instructions have already been sent.

val (tweetsAlreadyReplaced, tweetsToCheck) = inWindowCandidates

.partition(\_.entryWithItemIds.itemIds.exists(\_.head.entryIdToReplace.nonEmpty))

val tweetIdFromEntry: PartialFunction[PersistenceStoreEntry, Long] = {

case entry if entry.tweetId.nonEmpty => entry.tweetId.get

}

val tweetIdsAlreadyReplaced: Set[Long] = tweetsAlreadyReplaced.collect(tweetIdFromEntry).toSet

val tweetIdsToCheck: Seq[Long] = tweetsToCheck.collect(tweetIdFromEntry)

tweetIdsToCheck.filterNot(tweetIdsAlreadyReplaced.contains).distinct

} else Seq.empty

}

// The candidates here come from the Timelines Persistence Store, via a query feature

private def getApplicableCandidates(query: PipelineQuery): Seq[PersistenceStoreEntry] = {

val userAgent = UserAgent.fromString(query.clientContext.userAgent.getOrElse(""))

val clientPlatform = ClientPlatform.fromQueryOptions(query.clientContext.appId, userAgent)

val sortedResponses = query.features

.getOrElse(FeatureMap.empty)

.getOrElse(PersistenceEntriesFeature, Seq.empty)

.filter(\_.clientPlatform == clientPlatform)

.sortBy(-\_.servedTime.inMilliseconds)

val recentResponses = sortedResponses.indexWhere(\_.requestType == RequestType.Initial) match {

case -1 => sortedResponses

case lastGetInitialIndex => sortedResponses.take(lastGetInitialIndex + 1)

}

recentResponses.flatMap { r =>

r.entries.collect {

case entry if entry.entityIdType == EntityIdType.Tweet =>

PersistenceStoreEntry(entry, r.servedTime, r.clientPlatform, r.requestType)

}

}.distinct

}

}

case class PersistenceStoreEntry(

entryWithItemIds: EntryWithItemIds,

servedTime: Time,

clientPlatform: ClientPlatform,

requestType: RequestType) {

// Timelines Persistence Store currently includes 1 tweet ID per entryWithItemIds for tweets

val tweetId: Option[Long] = entryWithItemIds.itemIds.flatMap(\_.head.tweetId)

}