package com.twitter.home\_mixer.functional\_component.feature\_hydrator

import com.twitter.conversions.DurationOps.\_

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

import com.twitter.finagle.stats.StatsReceiver

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

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

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

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

import com.twitter.home\_mixer.model.request.FollowingProduct

import com.twitter.home\_mixer.model.request.ForYouProduct

import com.twitter.home\_mixer.service.HomeMixerAlertConfig

import com.twitter.product\_mixer.core.feature.Feature

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

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

import com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.QueryFeatureHydrator

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

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

import com.twitter.stitch.Stitch

import com.twitter.timelinemixer.clients.persistence.TimelineResponseBatchesClient

import com.twitter.timelinemixer.clients.persistence.TimelineResponseV3

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

import com.twitter.timelineservice.model.TimelineQuery

import com.twitter.timelineservice.model.core.TimelineKind

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

import com.twitter.util.Time

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

case class PersistenceStoreQueryFeatureHydrator @Inject() (

timelineResponseBatchesClient: TimelineResponseBatchesClient[TimelineResponseV3],

statsReceiver: StatsReceiver)

extends QueryFeatureHydrator[PipelineQuery] {

override val identifier: FeatureHydratorIdentifier = FeatureHydratorIdentifier("PersistenceStore")

private val scopedStatsReceiver = statsReceiver.scope(getClass.getSimpleName)

private val servedTweetIdsSizeStat = scopedStatsReceiver.stat("ServedTweetIdsSize")

private val WhoToFollowExcludedUserIdsLimit = 1000

private val ServedTweetIdsDuration = 10.minutes

private val ServedTweetIdsLimit = 100

private val ServedTweetPreviewIdsDuration = 10.hours

private val ServedTweetPreviewIdsLimit = 10

override val features: Set[Feature[\_, \_]] =

Set(

ServedTweetIdsFeature,

ServedTweetPreviewIdsFeature,

PersistenceEntriesFeature,

WhoToFollowExcludedUserIdsFeature)

private val supportedClients = Seq(

ClientPlatform.IPhone,

ClientPlatform.IPad,

ClientPlatform.Mac,

ClientPlatform.Android,

ClientPlatform.Web,

ClientPlatform.RWeb,

ClientPlatform.TweetDeckGryphon

)

override def hydrate(query: PipelineQuery): Stitch[FeatureMap] = {

val timelineKind = query.product match {

case FollowingProduct => TimelineKind.homeLatest

case ForYouProduct => TimelineKind.home

case other => throw new UnsupportedOperationException(s"Unknown product: $other")

}

val timelineQuery = TimelineQuery(id = query.getRequiredUserId, kind = timelineKind)

Stitch.callFuture {

timelineResponseBatchesClient

.get(query = timelineQuery, clientPlatforms = supportedClients)

.map { timelineResponses =>

// Note that the WTF entries are not being scoped by ClientPlatform

val whoToFollowUserIds = timelineResponses

.flatMap { timelineResponse =>

timelineResponse.entries

.filter(\_.entityIdType == EntityIdType.WhoToFollow)

.flatMap(\_.itemIds.toSeq.flatMap(\_.flatMap(\_.userId)))

}.take(WhoToFollowExcludedUserIdsLimit)

val clientPlatform = ClientPlatform.fromQueryOptions(

clientAppId = query.clientContext.appId,

userAgent = query.clientContext.userAgent.flatMap(UserAgent.fromString))

val servedTweetIds = timelineResponses

.filter(\_.clientPlatform == clientPlatform)

.filter(\_.servedTime >= Time.now - ServedTweetIdsDuration)

.sortBy(-\_.servedTime.inMilliseconds)

.flatMap(

\_.entries.flatMap(\_.tweetIds(includeSourceTweets = true)).take(ServedTweetIdsLimit))

servedTweetIdsSizeStat.add(servedTweetIds.size)

val servedTweetPreviewIds = timelineResponses

.filter(\_.clientPlatform == clientPlatform)

.filter(\_.servedTime >= Time.now - ServedTweetPreviewIdsDuration)

.sortBy(-\_.servedTime.inMilliseconds)

.flatMap(\_.entries

.filter(\_.entityIdType == EntityIdType.TweetPreview)

.flatMap(\_.tweetIds(includeSourceTweets = true)).take(ServedTweetPreviewIdsLimit))

FeatureMapBuilder()

.add(ServedTweetIdsFeature, servedTweetIds)

.add(ServedTweetPreviewIdsFeature, servedTweetPreviewIds)

.add(PersistenceEntriesFeature, timelineResponses)

.add(WhoToFollowExcludedUserIdsFeature, whoToFollowUserIds)

.build()

}

}

}

override val alerts = Seq(

HomeMixerAlertConfig.BusinessHours.defaultSuccessRateAlert(99.7, 50, 60, 60)

)

}