package com.twitter.interaction\_graph.scio.agg\_notifications

import com.spotify.scio.ScioMetrics

import com.twitter.clientapp.thriftscala.EventNamespace

import com.twitter.clientapp.thriftscala.LogEvent

import com.twitter.interaction\_graph.thriftscala.FeatureName

object InteractionGraphNotificationUtil {

val PUSH\_OPEN\_ACTIONS = Set("open", "background\_open")

val NTAB\_CLICK\_ACTIONS = Set("navigate", "click")

val STATUS\_ID\_REGEX = "^twitter:\\/\\/tweet\\?status\_id=([0-9]+).\*".r

val TWEET\_ID\_REGEX = "^twitter:\\/\\/tweet.id=([0-9]+).\*".r

def extractTweetIdFromUrl(url: String): Option[Long] = url match {

case STATUS\_ID\_REGEX(statusId) =>

ScioMetrics.counter("regex matching", "status\_id=").inc()

Some(statusId.toLong)

case TWEET\_ID\_REGEX(tweetId) =>

ScioMetrics.counter("regex matching", "tweet?id=").inc()

Some(tweetId.toLong)

case \_ => None

}

def getPushNtabEvents(e: LogEvent): Seq[(Long, (Long, FeatureName))] = {

for {

logBase <- e.logBase.toSeq

userId <- logBase.userId.toSeq

namespace <- e.eventNamespace.toSeq

(tweetId, featureName) <- namespace match {

case EventNamespace(\_, \_, \_, \_, \_, Some(action)) if PUSH\_OPEN\_ACTIONS.contains(action) =>

(for {

details <- e.eventDetails

url <- details.url

tweetId <- extractTweetIdFromUrl(url)

} yield {

ScioMetrics.counter("event type", "push open").inc()

(tweetId, FeatureName.NumPushOpens)

}).toSeq

case EventNamespace(\_, Some("ntab"), \_, \_, \_, Some("navigate")) =>

val tweetIds = for {

details <- e.eventDetails.toSeq

items <- details.items.toSeq

item <- items

ntabDetails <- item.notificationTabDetails.toSeq

clientEventMetadata <- ntabDetails.clientEventMetadata.toSeq

tweetIds <- clientEventMetadata.tweetIds.toSeq

tweetId <- tweetIds

} yield {

ScioMetrics.counter("event type", "ntab navigate").inc()

tweetId

}

tweetIds.map((\_, FeatureName.NumNtabClicks))

case EventNamespace(\_, Some("ntab"), \_, \_, \_, Some("click")) =>

val tweetIds = for {

details <- e.eventDetails.toSeq

items <- details.items.toSeq

item <- items

tweetId <- item.id

} yield {

ScioMetrics.counter("event type", "ntab click").inc()

tweetId

}

tweetIds.map((\_, FeatureName.NumNtabClicks))

case \_ => Nil

}

} yield (tweetId, (userId, featureName))

}

/\*\*

\* Returns events corresponding to ntab clicks. We have the tweet id from ntab clicks and can join

\* those with public tweets.

\*/

def getNtabEvents(e: LogEvent): Seq[(Long, (Long, FeatureName))] = {

for {

logBase <- e.logBase.toSeq

userId <- logBase.userId.toSeq

namespace <- e.eventNamespace.toSeq

(tweetId, featureName) <- namespace match {

case EventNamespace(\_, Some("ntab"), \_, \_, \_, Some("navigate")) =>

val tweetIds = for {

details <- e.eventDetails.toSeq

items <- details.items.toSeq

item <- items

ntabDetails <- item.notificationTabDetails.toSeq

clientEventMetadata <- ntabDetails.clientEventMetadata.toSeq

tweetIds <- clientEventMetadata.tweetIds.toSeq

tweetId <- tweetIds

} yield {

ScioMetrics.counter("event type", "ntab navigate").inc()

tweetId

}

tweetIds.map((\_, FeatureName.NumNtabClicks))

case EventNamespace(\_, Some("ntab"), \_, \_, \_, Some("click")) =>

val tweetIds = for {

details <- e.eventDetails.toSeq

items <- details.items.toSeq

item <- items

tweetId <- item.id

} yield {

ScioMetrics.counter("event type", "ntab click").inc()

tweetId

}

tweetIds.map((\_, FeatureName.NumNtabClicks))

case \_ => Nil

}

} yield (tweetId, (userId, featureName))

}

/\*\*

\* get push open events, keyed by impressionId (as the client event does not always have the tweetId nor the authorId)

\*/

def getPushOpenEvents(e: LogEvent): Seq[(String, (Long, FeatureName))] = {

for {

logBase <- e.logBase.toSeq

userId <- logBase.userId.toSeq

namespace <- e.eventNamespace.toSeq

(tweetId, featureName) <- namespace match {

case EventNamespace(\_, \_, \_, \_, \_, Some(action)) if PUSH\_OPEN\_ACTIONS.contains(action) =>

val impressionIdOpt = for {

details <- e.notificationDetails

impressionId <- details.impressionId

} yield {

ScioMetrics.counter("event type", "push open").inc()

impressionId

}

impressionIdOpt.map((\_, FeatureName.NumPushOpens)).toSeq

case \_ => Nil

}

} yield (tweetId, (userId, featureName))

}

}