package com.twitter.frigate.pushservice.util

import com.twitter.contentrecommender.thriftscala.MetricTag

import com.twitter.frigate.common.base.AlgorithmScore

import com.twitter.frigate.common.base.OutOfNetworkTweetCandidate

import com.twitter.frigate.common.base.SocialContextAction

import com.twitter.frigate.common.base.TopicCandidate

import com.twitter.frigate.common.base.TripCandidate

import com.twitter.frigate.pushservice.model.PushTypes.RawCandidate

import com.twitter.frigate.pushservice.model.PushTypes.Target

import com.twitter.frigate.thriftscala.{SocialContextAction => TSocialContextAction}

import com.twitter.frigate.thriftscala.{CommonRecommendationType => CRT}

import com.twitter.frigate.thriftscala.\_

import com.twitter.stitch.tweetypie.TweetyPie.TweetyPieResult

import com.twitter.topiclisting.utt.LocalizedEntity

import com.twitter.trends.trip\_v1.trip\_tweets.thriftscala.TripDomain

import scala.collection.Seq

case class MediaCRT(

crt: CRT,

photoCRT: CRT,

videoCRT: CRT)

object PushAdaptorUtil {

def getFrigateNotificationForUser(

crt: CRT,

userId: Long,

scActions: Seq[SocialContextAction],

pushCopyId: Option[Int],

ntabCopyId: Option[Int]

): FrigateNotification = {

val thriftSCActions = scActions.map { scAction =>

TSocialContextAction(

scAction.userId,

scAction.timestampInMillis,

scAction.tweetId

)

}

FrigateNotification(

crt,

NotificationDisplayLocation.PushToMobileDevice,

userNotification = Some(UserNotification(userId, thriftSCActions)),

pushCopyId = pushCopyId,

ntabCopyId = ntabCopyId

)

}

def getFrigateNotificationForTweet(

crt: CRT,

tweetId: Long,

scActions: Seq[TSocialContextAction],

authorIdOpt: Option[Long],

pushCopyId: Option[Int],

ntabCopyId: Option[Int],

simclusterId: Option[Int],

semanticCoreEntityIds: Option[List[Long]],

candidateContent: Option[CandidateContent],

trendId: Option[String],

tweetTripDomain: Option[scala.collection.Set[TripDomain]] = None

): FrigateNotification = {

FrigateNotification(

crt,

NotificationDisplayLocation.PushToMobileDevice,

tweetNotification = Some(

TweetNotification(

tweetId,

scActions,

authorIdOpt,

simclusterId,

semanticCoreEntityIds,

trendId,

tripDomain = tweetTripDomain)

),

pushCopyId = pushCopyId,

ntabCopyId = ntabCopyId,

candidateContent = candidateContent

)

}

def getFrigateNotificationForTweetWithSocialContextActions(

crt: CRT,

tweetId: Long,

scActions: Seq[SocialContextAction],

authorIdOpt: Option[Long],

pushCopyId: Option[Int],

ntabCopyId: Option[Int],

candidateContent: Option[CandidateContent],

semanticCoreEntityIds: Option[List[Long]],

trendId: Option[String]

): FrigateNotification = {

val thriftSCActions = scActions.map { scAction =>

TSocialContextAction(

scAction.userId,

scAction.timestampInMillis,

scAction.tweetId

)

}

getFrigateNotificationForTweet(

crt = crt,

tweetId = tweetId,

scActions = thriftSCActions,

authorIdOpt = authorIdOpt,

pushCopyId = pushCopyId,

ntabCopyId = ntabCopyId,

simclusterId = None,

candidateContent = candidateContent,

semanticCoreEntityIds = semanticCoreEntityIds,

trendId = trendId

)

}

def generateOutOfNetworkTweetCandidates(

inputTarget: Target,

id: Long,

mediaCRT: MediaCRT,

result: Option[TweetyPieResult],

localizedEntity: Option[LocalizedEntity] = None,

isMrBackfillFromCR: Option[Boolean] = None,

tagsFromCR: Option[Seq[MetricTag]] = None,

score: Option[Double] = None,

algorithmTypeCR: Option[String] = None,

tripTweetDomain: Option[scala.collection.Set[TripDomain]] = None

): RawCandidate

with OutOfNetworkTweetCandidate

with TopicCandidate

with TripCandidate

with AlgorithmScore = {

new RawCandidate

with OutOfNetworkTweetCandidate

with TopicCandidate

with TripCandidate

with AlgorithmScore {

override val tweetId: Long = id

override val target: Target = inputTarget

override val tweetyPieResult: Option[TweetyPieResult] = result

override val localizedUttEntity: Option[LocalizedEntity] = localizedEntity

override val semanticCoreEntityId: Option[Long] = localizedEntity.map(\_.entityId)

override def commonRecType: CRT =

getMediaBasedCRT(mediaCRT.crt, mediaCRT.photoCRT, mediaCRT.videoCRT)

override def isMrBackfillCR: Option[Boolean] = isMrBackfillFromCR

override def tagsCR: Option[Seq[MetricTag]] = tagsFromCR

override def algorithmScore: Option[Double] = score

override def algorithmCR: Option[String] = algorithmTypeCR

override def tripDomain: Option[collection.Set[TripDomain]] = tripTweetDomain

}

}

}