package com.twitter.frigate.pushservice.model.ibis

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

import com.twitter.frigate.pushservice.model.MagicFanoutEventHydratedCandidate

import com.twitter.frigate.pushservice.params.PushConstants

import com.twitter.frigate.pushservice.params.PushFeatureSwitchParams

import com.twitter.frigate.pushservice.predicate.magic\_fanout.MagicFanoutPredicatesUtil

import com.twitter.frigate.pushservice.util.PushIbisUtil.\_

import com.twitter.util.Future

trait MagicFanoutNewsEventIbis2Hydrator extends Ibis2HydratorForCandidate {

self: PushCandidate with MagicFanoutEventHydratedCandidate =>

override lazy val senderId: Option[Long] = {

val isUgmMoment = self.semanticCoreEntityTags.values.flatten.toSet

.contains(MagicFanoutPredicatesUtil.UgmMomentTag)

owningTwitterUserIds.headOption match {

case Some(owningTwitterUserId)

if isUgmMoment && target.params(

PushFeatureSwitchParams.MagicFanoutNewsUserGeneratedEventsEnable) =>

Some(owningTwitterUserId)

case \_ => None

}

}

lazy val stats = self.statsReceiver.scope("MagicFanout")

lazy val defaultImageCounter = stats.counter("default\_image")

lazy val requestImageCounter = stats.counter("request\_num")

lazy val noneImageCounter = stats.counter("none\_num")

private def getModelValueMediaUrl(

urlOpt: Option[String],

mapKey: String

): Option[(String, String)] = {

requestImageCounter.incr()

urlOpt match {

case Some(PushConstants.DefaultEventMediaUrl) =>

defaultImageCounter.incr()

None

case Some(url) => Some(mapKey -> url)

case None =>

noneImageCounter.incr()

None

}

}

private lazy val eventModelValuesFut: Future[Map[String, String]] = {

for {

title <- eventTitleFut

squareImageUrl <- squareImageUrlFut

primaryImageUrl <- primaryImageUrlFut

eventDescriptionOpt <- eventDescriptionFut

} yield {

val authorId = owningTwitterUserIds.headOption match {

case Some(author)

if target.params(PushFeatureSwitchParams.MagicFanoutNewsUserGeneratedEventsEnable) =>

Some("author" -> author.toString)

case \_ => None

}

val eventDescription = eventDescriptionOpt match {

case Some(description)

if target.params(PushFeatureSwitchParams.MagicFanoutNewsEnableDescriptionCopy) =>

Some("event\_description" -> description)

case \_ =>

None

}

Map(

"event\_id" -> s"$eventId",

"event\_title" -> title

) ++

getModelValueMediaUrl(squareImageUrl, "square\_media\_url") ++

getModelValueMediaUrl(primaryImageUrl, "media\_url") ++

authorId ++

eventDescription

}

}

private lazy val topicValuesFut: Future[Map[String, String]] = {

if (target.params(PushFeatureSwitchParams.EnableTopicCopyForMF)) {

followedTopicLocalizedEntities.map(\_.headOption).flatMap {

case Some(localizedEntity) =>

Future.value(Map("topic\_name" -> localizedEntity.localizedNameForDisplay))

case \_ =>

ergLocalizedEntities.map(\_.headOption).map {

case Some(localizedEntity)

if target.params(PushFeatureSwitchParams.EnableTopicCopyForImplicitTopics) =>

Map("topic\_name" -> localizedEntity.localizedNameForDisplay)

case \_ => Map.empty[String, String]

}

}

} else {

Future.value(Map.empty[String, String])

}

}

override lazy val modelValues: Future[Map[String, String]] =

mergeFutModelValues(super.modelValues, mergeFutModelValues(eventModelValuesFut, topicValuesFut))

}