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

import com.twitter.frigate.common.base.BaseGameScore

import com.twitter.frigate.common.base.MagicFanoutSportsEventCandidate

import com.twitter.frigate.common.base.MagicFanoutSportsScoreInformation

import com.twitter.frigate.common.base.NflGameScore

import com.twitter.frigate.common.base.SoccerGameScore

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.predicate.magic\_fanout.MagicFanoutSportsUtil

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

import com.twitter.util.Future

trait MagicFanoutSportsEventIbis2Hydrator extends Ibis2HydratorForCandidate {

self: PushCandidate

with MagicFanoutEventHydratedCandidate

with MagicFanoutSportsEventCandidate

with MagicFanoutSportsScoreInformation =>

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

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

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

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

override lazy val relevanceScoreMapFut = Future.value(Map.empty[String, String])

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

} yield {

Map(

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

"event\_title" -> title

) ++

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

getModelValueMediaUrl(primaryImageUrl, "media\_url")

}

}

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

for {

scores <- gameScores

homeName <- homeTeamInfo.map(\_.map(\_.name))

awayName <- awayTeamInfo.map(\_.map(\_.name))

} yield {

if (awayName.isDefined && homeName.isDefined && scores.isDefined) {

scores.get match {

case game: SoccerGameScore =>

MagicFanoutSportsUtil.getSoccerIbisMap(game) ++ Map(

"away\_team" -> awayName.get,

"home\_team" -> homeName.get

)

case game: NflGameScore =>

MagicFanoutSportsUtil.getNflIbisMap(game) ++ Map(

"away\_team" -> MagicFanoutSportsUtil.getNFLReadableName(awayName.get),

"home\_team" -> MagicFanoutSportsUtil.getNFLReadableName(homeName.get)

)

case baseGameScore: BaseGameScore =>

Map.empty[String, String]

}

} else Map.empty[String, String]

}

}

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

mergeFutModelValues(super.customFieldsMapFut, sportsScoreValues)

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

mergeFutModelValues(super.modelValues, eventModelValuesFut)

}