package com.twitter.frigate.pushservice.send\_handler.generator

import com.twitter.datatools.entityservice.entities.sports.thriftscala.BaseballGameLiveUpdate

import com.twitter.datatools.entityservice.entities.sports.thriftscala.BasketballGameLiveUpdate

import com.twitter.datatools.entityservice.entities.sports.thriftscala.CricketMatchLiveUpdate

import com.twitter.datatools.entityservice.entities.sports.thriftscala.NflFootballGameLiveUpdate

import com.twitter.datatools.entityservice.entities.sports.thriftscala.SoccerMatchLiveUpdate

import com.twitter.escherbird.common.thriftscala.Domains

import com.twitter.escherbird.common.thriftscala.QualifiedId

import com.twitter.escherbird.metadata.thriftscala.EntityMegadata

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.TeamInfo

import com.twitter.frigate.magic\_events.thriftscala.MagicEventsReason

import com.twitter.frigate.pushservice.exception.InvalidSportDomainException

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

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

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

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

import com.twitter.frigate.thriftscala.CommonRecommendationType

import com.twitter.frigate.thriftscala.FrigateNotification

import com.twitter.frigate.thriftscala.MagicFanoutEventNotificationDetails

import com.twitter.hermit.store.semantic\_core.SemanticEntityForQuery

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Future

object MagicFanoutSportsEventCandidateGenerator {

final def getCandidate(

targetUser: Target,

notification: FrigateNotification,

basketballGameScoreStore: ReadableStore[QualifiedId, BasketballGameLiveUpdate],

baseballGameScoreStore: ReadableStore[QualifiedId, BaseballGameLiveUpdate],

cricketMatchScoreStore: ReadableStore[QualifiedId, CricketMatchLiveUpdate],

soccerMatchScoreStore: ReadableStore[QualifiedId, SoccerMatchLiveUpdate],

nflGameScoreStore: ReadableStore[QualifiedId, NflFootballGameLiveUpdate],

semanticCoreMegadataStore: ReadableStore[SemanticEntityForQuery, EntityMegadata],

): Future[RawCandidate] = {

/\*\*

\* frigateNotification recommendation type should be [[CommonRecommendationType.MagicFanoutSportsEvent]]

\* AND pushId field should be set

\*

\* \*/

require(

notification.commonRecommendationType == CommonRecommendationType.MagicFanoutSportsEvent,

"MagicFanoutSports: unexpected CRT " + notification.commonRecommendationType

)

require(

notification.magicFanoutEventNotification.exists(\_.pushId.isDefined),

"MagicFanoutSportsEvent: pushId is not defined")

val magicFanoutEventNotification = notification.magicFanoutEventNotification.get

val eventId = magicFanoutEventNotification.eventId

val \_isScoreUpdate = magicFanoutEventNotification.isScoreUpdate.getOrElse(false)

val gameScoresFut: Future[Option[BaseGameScore]] = {

if (\_isScoreUpdate) {

semanticCoreMegadataStore

.get(SemanticEntityForQuery(PushConstants.SportsEventDomainId, eventId))

.flatMap {

case Some(megadata) =>

if (megadata.domains.contains(Domains.BasketballGame)) {

basketballGameScoreStore

.get(QualifiedId(Domains.BasketballGame.value, eventId)).map {

case Some(game) if game.status.isDefined =>

val status = game.status.get

MagicFanoutSportsUtil.transformToGameScore(game.score, status)

case \_ => None

}

} else if (megadata.domains.contains(Domains.BaseballGame)) {

baseballGameScoreStore

.get(QualifiedId(Domains.BaseballGame.value, eventId)).map {

case Some(game) if game.status.isDefined =>

val status = game.status.get

MagicFanoutSportsUtil.transformToGameScore(game.runs, status)

case \_ => None

}

} else if (megadata.domains.contains(Domains.NflFootballGame)) {

nflGameScoreStore

.get(QualifiedId(Domains.NflFootballGame.value, eventId)).map {

case Some(game) if game.status.isDefined =>

val nflScore = MagicFanoutSportsUtil.transformNFLGameScore(game)

nflScore

case \_ => None

}

} else if (megadata.domains.contains(Domains.SoccerMatch)) {

soccerMatchScoreStore

.get(QualifiedId(Domains.SoccerMatch.value, eventId)).map {

case Some(game) if game.status.isDefined =>

val soccerScore = MagicFanoutSportsUtil.transformSoccerGameScore(game)

soccerScore

case \_ => None

}

} else {

// The domains are not in our list of supported sports

throw new InvalidSportDomainException(

s"Domain for entity ${eventId} is not supported")

}

case \_ => Future.None

}

} else Future.None

}

val homeTeamInfoFut: Future[Option[TeamInfo]] = gameScoresFut.flatMap {

case Some(gameScore) =>

MagicFanoutSportsUtil.getTeamInfo(gameScore.home, semanticCoreMegadataStore)

case \_ => Future.None

}

val awayTeamInfoFut: Future[Option[TeamInfo]] = gameScoresFut.flatMap {

case Some(gameScore) =>

MagicFanoutSportsUtil.getTeamInfo(gameScore.away, semanticCoreMegadataStore)

case \_ => Future.None

}

val candidate = new RawCandidate

with MagicFanoutSportsEventCandidate

with MagicFanoutSportsScoreInformation {

override val target: Target = targetUser

override val eventId: Long = magicFanoutEventNotification.eventId

override val pushId: Long = magicFanoutEventNotification.pushId.get

override val candidateMagicEventsReasons: Seq[MagicEventsReason] =

magicFanoutEventNotification.eventReasons.getOrElse(Seq.empty)

override val momentId: Option[Long] = magicFanoutEventNotification.momentId

override val eventLanguage: Option[String] = magicFanoutEventNotification.eventLanguage

override val details: Option[MagicFanoutEventNotificationDetails] =

magicFanoutEventNotification.details

override val frigateNotification: FrigateNotification = notification

override val homeTeamInfo: Future[Option[TeamInfo]] = homeTeamInfoFut

override val awayTeamInfo: Future[Option[TeamInfo]] = awayTeamInfoFut

override val gameScores: Future[Option[BaseGameScore]] = gameScoresFut

override val isScoreUpdate: Boolean = \_isScoreUpdate

}

Future.value(candidate)

}

}