package com.twitter.frigate.pushservice.model.ntab

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

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

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

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

import com.twitter.frigate.pushservice.params.{PushFeatureSwitchParams => FS}

import com.twitter.notificationservice.thriftscala.CreateGenericNotificationRequest

import com.twitter.notificationservice.thriftscala.DisplayText

import com.twitter.notificationservice.thriftscala.DisplayTextEntity

import com.twitter.notificationservice.thriftscala.GenericType

import com.twitter.notificationservice.thriftscala.TextValue

import com.twitter.notificationservice.thriftscala.TapThroughAction

import com.twitter.util.Future

import com.twitter.util.Time

trait MagicFanoutSportsEventNTabRequestHydrator extends EventNTabRequestHydrator {

self: PushCandidate

with MagicFanoutEventHydratedCandidate

with MagicFanoutSportsEventCandidate

with MagicFanoutSportsScoreInformation =>

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

lazy val inNetworkOnlyCounter = stats.counter("in\_network\_only")

lazy val facePilesEnabledCounter = stats.counter("face\_piles\_enabled")

lazy val facePilesDisabledCounter = stats.counter("face\_piles\_disabled")

lazy val filterPeopleWhoDontFollowMeCounter = stats.counter("pepole\_who\_dont\_follow\_me\_counter")

override lazy val tapThroughFut: Future[String] = {

Future.value(s"i/events/$eventId")

}

override lazy val displayTextEntitiesFut: Future[Seq[DisplayTextEntity]] =

eventTitleFut.map { eventTitle =>

Seq(DisplayTextEntity(name = "title", value = TextValue.Text(eventTitle)))

}

override lazy val facepileUsersFut: Future[Seq[Long]] =

if (target.params(FS.EnableNTabFacePileForSportsEventNotifications)) {

Future

.join(

target.notificationsFromOnlyPeopleIFollow,

target.filterNotificationsFromPeopleThatDontFollowMe,

awayTeamInfo,

homeTeamInfo).map {

case (inNetworkOnly, filterPeopleWhoDontFollowMe, away, home)

if !(inNetworkOnly || filterPeopleWhoDontFollowMe) =>

val awayTeamId = away.flatMap(\_.twitterUserId)

val homeTeamId = home.flatMap(\_.twitterUserId)

facePilesEnabledCounter.incr

Seq(awayTeamId, homeTeamId).flatten

case (inNetworkOnly, filterPeopleWhoDontFollowMe, \_, \_) =>

facePilesDisabledCounter.incr

if (inNetworkOnly) inNetworkOnlyCounter.incr

if (filterPeopleWhoDontFollowMe) filterPeopleWhoDontFollowMeCounter.incr

Seq.empty[Long]

}

} else Future.Nil

private lazy val sportsNtabRequest: Future[Option[CreateGenericNotificationRequest]] = {

Future

.join(senderIdFut, displayTextEntitiesFut, facepileUsersFut, tapThroughFut)

.map {

case (senderId, displayTextEntities, facepileUsers, tapThrough) =>

Some(

CreateGenericNotificationRequest(

userId = target.targetId,

senderId = senderId,

genericType = GenericType.RefreshableNotification,

displayText = DisplayText(values = displayTextEntities),

facepileUsers = facepileUsers,

timestampMillis = Time.now.inMillis,

tapThroughAction = Some(TapThroughAction(Some(tapThrough))),

impressionId = Some(impressionId),

socialProofText = socialProofDisplayText,

context = storyContext,

inlineCard = inlineCard,

refreshableType = refreshableType

))

}

}

override lazy val ntabRequest: Future[Option[CreateGenericNotificationRequest]] = {

if (target.params(FS.EnableNTabEntriesForSportsEventNotifications)) {

self.target.history.flatMap { pushHistory =>

val prevEventHistoryExists = pushHistory.sortedHistory.exists {

case (\_, notification) =>

notification.magicFanoutEventNotification.exists(\_.eventId == self.eventId)

}

if (prevEventHistoryExists) {

Future.None

} else sportsNtabRequest

}

} else Future.None

}

}