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

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

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

import com.twitter.frigate.pushservice.model.MagicFanoutCreatorEventPushCandidate

import com.twitter.frigate.pushservice.take.NotificationServiceSender

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.InlineCard

import com.twitter.notificationservice.thriftscala.StoryContext

import com.twitter.notificationservice.thriftscala.TextValue

import com.twitter.notificationservice.thriftscala.TapThroughAction

import com.twitter.util.Future

import com.twitter.util.Time

trait MagicFanoutCreatorEventNtabRequestHydrator extends NTabRequestHydrator {

self: PushCandidate with MagicFanoutCreatorEventPushCandidate =>

override val senderIdFut: Future[Long] = Future.value(creatorId)

override lazy val tapThroughFut: Future[String] =

Future.value(s"/${userProfile.screenName}/superfollows/subscribe")

lazy val optionalTweetCountEntityFut: Future[Option[DisplayTextEntity]] = {

creatorFanoutType match {

case CreatorFanoutType.UserSubscription =>

numberOfTweetsFut.map {

\_.flatMap {

case numberOfTweets if numberOfTweets >= 10 =>

Some(

DisplayTextEntity(

name = "tweet\_count",

emphasis = true,

value = TextValue.Text(numberOfTweets.toString)))

case \_ => None

}

}

case \_ => Future.None

}

}

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

optionalTweetCountEntityFut

.map { tweetCountOpt =>

Seq(

NotificationServiceSender

.getDisplayTextEntityFromUser(hydratedCreator, "display\_name", isBold = true),

tweetCountOpt).flatten

}

override lazy val facepileUsersFut: Future[Seq[Long]] = Future.value(Seq(creatorId))

override val storyContext: Option[StoryContext] = None

override val inlineCard: Option[InlineCard] = None

lazy val refreshableTypeFut = {

creatorFanoutType match {

case CreatorFanoutType.UserSubscription =>

numberOfTweetsFut.map {

\_.flatMap {

case numberOfTweets if numberOfTweets >= 10 =>

Some("MagicFanoutCreatorSubscriptionWithTweets")

case \_ => super.refreshableType

}

}

case \_ => Future.value(super.refreshableType)

}

}

override lazy val socialProofDisplayText: Option[DisplayText] = {

creatorFanoutType match {

case CreatorFanoutType.UserSubscription =>

Some(

DisplayText(values = Seq(

DisplayTextEntity(name = "handle", value = TextValue.Text(userProfile.screenName)))))

case CreatorFanoutType.NewCreator => None

case \_ => None

}

}

override lazy val ntabRequest = {

Future

.join(

senderIdFut,

displayTextEntitiesFut,

facepileUsersFut,

tapThroughFut,

refreshableTypeFut).map {

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

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 = refreshableTypeOpt

))

}

}

}