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

import com.twitter.frigate.common.base.MagicFanoutProductLaunchCandidate

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

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

import com.twitter.notificationservice.thriftscala.\_

import com.twitter.util.Future

import com.twitter.util.Time

trait MagicFanoutProductLaunchNtabRequestHydrator extends NTabRequestHydrator {

self: PushCandidate with MagicFanoutProductLaunchCandidate =>

override val senderIdFut: Future[Long] = Future.value(0L)

override lazy val tapThroughFut: Future[String] = Future.value(getProductLaunchTapThrough())

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

Future.value(

frigateNotification.magicFanoutProductLaunchNotification

.flatMap {

\_.productInfo.flatMap {

\_.body.map { body =>

Seq(

DisplayTextEntity(name = "body", value = TextValue.Text(body)),

)

}

}

}.getOrElse(Nil))

}

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

Future.value(

frigateNotification.magicFanoutProductLaunchNotification

.flatMap {

\_.productInfo.flatMap {

\_.facepileUsers

}

}.getOrElse(Nil))

}

override val storyContext: Option[StoryContext] = None

override val inlineCard: Option[InlineCard] = None

override lazy val socialProofDisplayText: Option[DisplayText] = {

frigateNotification.magicFanoutProductLaunchNotification.flatMap {

\_.productInfo.flatMap {

\_.title.map { title =>

DisplayText(values =

Seq(DisplayTextEntity(name = "social\_context", value = TextValue.Text(title))))

}

}

}

}

lazy val defaultTapThrough = target.params(PushFeatureSwitchParams.ProductLaunchTapThrough)

private def getProductLaunchTapThrough(): String = {

frigateNotification.magicFanoutProductLaunchNotification match {

case Some(productLaunchNotif) =>

productLaunchNotif.productInfo match {

case Some(productInfo) => productInfo.tapThrough.getOrElse(defaultTapThrough)

case \_ => defaultTapThrough

}

case \_ => defaultTapThrough

}

}

private lazy val productLaunchNtabRequest: 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(PushFeatureSwitchParams.EnableNTabEntriesForProductLaunchNotifications)) {

productLaunchNtabRequest

} else Future.None

}

}