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

import com.twitter.frigate.common.base.SpaceCandidate

import com.twitter.frigate.common.util.MrNtabCopyObjects

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

import com.twitter.frigate.pushservice.model.ScheduledSpaceSpeakerPushCandidate

import com.twitter.frigate.pushservice.model.ScheduledSpaceSubscriberPushCandidate

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

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

import com.twitter.frigate.thriftscala.SpaceNotificationType

import com.twitter.gizmoduck.thriftscala.User

import com.twitter.notificationservice.thriftscala.\_

import com.twitter.util.Future

import com.twitter.util.Time

trait ScheduledSpaceSpeakerNTabRequestHydrator extends ScheduledSpaceNTabRequestHydrator {

self: PushCandidate with ScheduledSpaceSpeakerPushCandidate =>

override def refreshableType: Option[String] = {

frigateNotification.spaceNotification.flatMap { spaceNotification =>

spaceNotification.spaceNotificationType.flatMap {

case SpaceNotificationType.PreSpaceBroadcast =>

MrNtabCopyObjects.ScheduledSpaceSpeakerSoon.refreshableType

case SpaceNotificationType.AtSpaceBroadcast =>

MrNtabCopyObjects.ScheduledSpaceSpeakerNow.refreshableType

case \_ =>

throw new IllegalStateException(s"Unexpected SpaceNotificationType")

}

}

}

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

override val socialProofDisplayText: Option[DisplayText] = Some(DisplayText())

}

trait ScheduledSpaceSubscriberNTabRequestHydrator extends ScheduledSpaceNTabRequestHydrator {

self: PushCandidate with ScheduledSpaceSubscriberPushCandidate =>

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

hostId match {

case Some(spaceHostId) => Future.value(Seq(spaceHostId))

case \_ =>

Future.exception(

new IllegalStateException(

"Unable to get host id for ScheduledSpaceSubscriberNTabRequestHydrator"))

}

}

override val socialProofDisplayText: Option[DisplayText] = None

}

trait ScheduledSpaceNTabRequestHydrator extends NTabRequestHydrator {

self: PushCandidate with SpaceCandidate =>

def hydratedHost: Option[User]

override lazy val senderIdFut: Future[Long] = {

hostId match {

case Some(spaceHostId) => Future.value(spaceHostId)

case \_ => throw new IllegalStateException(s"No Space Host Id")

}

}

override lazy val tapThroughFut: Future[String] = Future.value(s"i/spaces/$spaceId")

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

NotificationServiceSender

.getDisplayTextEntityFromUser(

Future.value(hydratedHost),

fieldName = "space\_host\_name",

isBold = true

).map(\_.toSeq)

override val storyContext: Option[StoryContext] = None

override val inlineCard: Option[InlineCard] = None

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

Future.join(senderIdFut, displayTextEntitiesFut, facepileUsersFut, tapThroughFut).map {

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

val expiryTimeMillis = if (target.params(PushFeatureSwitchParams.EnableSpacesTtlForNtab)) {

Some(

(Time.now + target.params(

PushFeatureSwitchParams.SpaceNotificationsTTLDurationForNTab)).inMillis)

} else None

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,

expiryTimeMillis = expiryTimeMillis

))

}

}

}