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

import com.twitter.frigate.common.base.TopicCandidate

import com.twitter.frigate.common.base.TweetAuthorDetails

import com.twitter.frigate.common.base.TweetCandidate

import com.twitter.frigate.common.base.TweetDetails

import com.twitter.frigate.common.rec\_types.RecTypes.\_

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

import com.twitter.frigate.pushservice.exception.TweetNTabRequestHydratorException

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

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

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

import com.twitter.notificationservice.thriftscala.DisplayText

import com.twitter.notificationservice.thriftscala.DisplayTextEntity

import com.twitter.notificationservice.thriftscala.TextValue

import com.twitter.util.Future

trait OutOfNetworkTweetNTabRequestHydrator extends TweetNTabRequestHydrator {

self: PushCandidate

with TweetCandidate

with TweetAuthorDetails

with TopicCandidate

with TweetDetails =>

lazy val useTopicCopyForMBCGNtab = mrModelingBasedTypes.contains(commonRecType) && target.params(

PushFeatureSwitchParams.EnableMrModelingBasedCandidatesTopicCopy)

lazy val useTopicCopyForFrsNtab = frsTypes.contains(commonRecType) && target.params(

PushFeatureSwitchParams.EnableFrsTweetCandidatesTopicCopy)

lazy val useTopicCopyForTagspaceNtab = tagspaceTypes.contains(commonRecType) && target.params(

PushFeatureSwitchParams.EnableHashspaceCandidatesTopicCopy)

override lazy val tapThroughFut: Future[String] = {

if (hasVideo && self.target.params(

PushFeatureSwitchParams.EnableLaunchVideosInImmersiveExplore)) {

Future.value(

s"i/immersive\_timeline?display\_location=notification&include\_pinned\_tweet=true&pinned\_tweet\_id=${tweetId}&tl\_type=imv")

} else {

tweetAuthor.map {

case Some(author) =>

val authorProfile = author.profile.getOrElse(

throw new TweetNTabRequestHydratorException(

s"Unable to obtain author profile for: ${author.id}"))

s"${authorProfile.screenName}/status/${tweetId.toString}"

case \_ =>

throw new TweetNTabRequestHydratorException(

s"Unable to obtain author and target details to generate tap through for Tweet: $tweetId")

}

}

}

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

if (localizedUttEntity.isDefined &&

(useTopicCopyForMBCGNtab || useTopicCopyForFrsNtab || useTopicCopyForTagspaceNtab)) {

NotificationServiceSender

.getDisplayTextEntityFromUser(tweetAuthor, "tweetAuthorName", isBold = true).map(\_.toSeq)

} else {

NotificationServiceSender

.getDisplayTextEntityFromUser(tweetAuthor, "author", isBold = true).map(\_.toSeq)

}

override lazy val refreshableType: Option[String] = {

if (localizedUttEntity.isDefined &&

(useTopicCopyForMBCGNtab || useTopicCopyForFrsNtab || useTopicCopyForTagspaceNtab)) {

MrNtabCopyObjects.TopicTweet.refreshableType

} else ntabCopy.refreshableType

}

override def socialProofDisplayText: Option[DisplayText] = {

if (localizedUttEntity.isDefined &&

(useTopicCopyForMBCGNtab || useTopicCopyForFrsNtab || useTopicCopyForTagspaceNtab)) {

localizedUttEntity.map(uttEntity =>

DisplayText(values =

Seq(DisplayTextEntity("topic\_name", TextValue.Text(uttEntity.localizedNameForDisplay)))))

} else None

}

override lazy val facepileUsersFut: Future[Seq[Long]] = senderIdFut.map(Seq(\_))

}