package com.twitter.home\_mixer.functional\_component.decorator.urt.builder

import com.twitter.home\_mixer.model.HomeFeatures.FocalTweetAuthorIdFeature

import com.twitter.home\_mixer.model.HomeFeatures.FocalTweetInNetworkFeature

import com.twitter.home\_mixer.model.HomeFeatures.FocalTweetRealNamesFeature

import com.twitter.home\_mixer.model.HomeFeatures.InNetworkFeature

import com.twitter.home\_mixer.product.following.model.HomeMixerExternalStrings

import com.twitter.product\_mixer.component\_library.model.candidate.TweetCandidate

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.functional\_component.decorator.urt.builder.social\_context.BaseSocialContextBuilder

import com.twitter.product\_mixer.core.model.marshalling.response.urt.metadata.SocialContext

import com.twitter.product\_mixer.core.model.marshalling.response.urt.metadata.\_

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

import com.twitter.product\_mixer.core.product.guice.scope.ProductScoped

import com.twitter.stringcenter.client.StringCenter

import javax.inject.Inject

import javax.inject.Provider

import javax.inject.Singleton

/\*\*

\* Use '@A replied' when the root tweet is out-of-network and the reply is in network.

\*

\* This function should only be called for the root Tweet of convo modules. This is enforced by

\* [[HomeTweetSocialContextBuilder]].

\*/

@Singleton

case class ExtendedReplySocialContextBuilder @Inject() (

externalStrings: HomeMixerExternalStrings,

@ProductScoped stringCenterProvider: Provider[StringCenter])

extends BaseSocialContextBuilder[PipelineQuery, TweetCandidate] {

private val stringCenter = stringCenterProvider.get()

private val extendedReplyString = externalStrings.socialContextExtendedReply

def apply(

query: PipelineQuery,

candidate: TweetCandidate,

candidateFeatures: FeatureMap

): Option[SocialContext] = {

// If these values are missing default to not showing an extended reply banner

val inNetworkRoot = candidateFeatures.getOrElse(InNetworkFeature, true)

val inNetworkFocalTweet =

candidateFeatures.getOrElse(FocalTweetInNetworkFeature, None).getOrElse(false)

if (!inNetworkRoot && inNetworkFocalTweet) {

val focalTweetAuthorIdOpt = candidateFeatures.getOrElse(FocalTweetAuthorIdFeature, None)

val focalTweetRealNames =

candidateFeatures

.getOrElse(FocalTweetRealNamesFeature, None).getOrElse(Map.empty[Long, String])

val focalTweetAuthorNameOpt = focalTweetAuthorIdOpt.flatMap(focalTweetRealNames.get)

(focalTweetAuthorIdOpt, focalTweetAuthorNameOpt) match {

case (Some(focalTweetAuthorId), Some(focalTweetAuthorName)) =>

Some(

GeneralContext(

contextType = ConversationGeneralContextType,

text = stringCenter

.prepare(extendedReplyString, placeholders = Map("user1" -> focalTweetAuthorName)),

url = None,

contextImageUrls = None,

landingUrl = Some(

Url(

urlType = DeepLink,

url = "",

urtEndpointOptions = None

))

))

case \_ =>

None

}

} else {

None

}

}

}