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

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

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

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

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

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 received a reply' as social context when the root Tweet is in network and the focal tweet is OON.

\*

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

\* [[HomeTweetSocialContextBuilder]].

\*/

@Singleton

case class ReceivedReplySocialContextBuilder @Inject() (

externalStrings: HomeMixerExternalStrings,

@ProductScoped stringCenterProvider: Provider[StringCenter])

extends BaseSocialContextBuilder[PipelineQuery, TweetCandidate] {

private val stringCenter = stringCenterProvider.get()

private val receivedReplyString = externalStrings.socialContextReceivedReply

def apply(

query: PipelineQuery,

candidate: TweetCandidate,

candidateFeatures: FeatureMap

): Option[SocialContext] = {

// If these values are missing default to not showing a received a reply banner

val inNetwork = candidateFeatures.getOrElse(InNetworkFeature, false)

val inNetworkFocalTweet =

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

if (inNetwork && !inNetworkFocalTweet) {

val authorIdOpt = candidateFeatures.getOrElse(AuthorIdFeature, None)

val realNames = candidateFeatures.getOrElse(RealNamesFeature, Map.empty[Long, String])

val authorNameOpt = authorIdOpt.flatMap(realNames.get)

(authorIdOpt, authorNameOpt) match {

case (Some(authorId), Some(authorName)) =>

Some(

GeneralContext(

contextType = ConversationGeneralContextType,

text = stringCenter

.prepare(receivedReplyString, placeholders = Map("user1" -> authorName)),

url = None,

contextImageUrls = None,

landingUrl = Some(

Url(

urlType = DeepLink,

url = "",

urtEndpointOptions = None

)

)

)

)

case \_ => None

}

} else {

None

}

}

}