package com.twitter.home\_mixer.functional\_component.selector

import com.twitter.home\_mixer.functional\_component.selector.UpdateNewTweetsPillDecoration.NumAvatars

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

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

import com.twitter.home\_mixer.model.request.HasDeviceContext

import com.twitter.home\_mixer.param.HomeGlobalParams.EnableNewTweetsPillAvatarsParam

import com.twitter.home\_mixer.util.CandidatesUtil

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

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

import com.twitter.product\_mixer.component\_library.model.presentation.urt.UrtItemPresentation

import com.twitter.product\_mixer.core.functional\_component.common.CandidateScope

import com.twitter.product\_mixer.core.functional\_component.selector.Selector

import com.twitter.product\_mixer.core.functional\_component.selector.SelectorResult

import com.twitter.product\_mixer.core.model.common.presentation.CandidateWithDetails

import com.twitter.product\_mixer.core.model.common.presentation.ItemCandidateWithDetails

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

import com.twitter.product\_mixer.core.model.marshalling.response.urt.richtext.RichText

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

import com.twitter.stringcenter.client.StringCenter

import com.twitter.stringcenter.client.core.ExternalString

object UpdateNewTweetsPillDecoration {

val NumAvatars = 3

}

case class UpdateNewTweetsPillDecoration[Query <: PipelineQuery with HasDeviceContext](

override val pipelineScope: CandidateScope,

stringCenter: StringCenter,

seeNewTweetsString: ExternalString,

tweetedString: ExternalString)

extends Selector[Query] {

override def apply(

query: Query,

remainingCandidates: Seq[CandidateWithDetails],

result: Seq[CandidateWithDetails]

): SelectorResult = {

val (alerts, otherCandidates) =

remainingCandidates.partition(candidate =>

candidate.isCandidateType[ShowAlertCandidate]() && pipelineScope.contains(candidate))

val updatedCandidates = alerts

.collectFirst {

case newTweetsPill: ItemCandidateWithDetails =>

val userIds = CandidatesUtil

.getItemCandidatesWithOnlyModuleLast(result)

.filter(candidate =>

candidate.isCandidateType[TweetCandidate]() && pipelineScope.contains(candidate))

.filterNot(\_.features.getOrElse(IsRetweetFeature, false))

.flatMap(\_.features.getOrElse(AuthorIdFeature, None))

.filterNot(\_ == query.getRequiredUserId)

.distinct

val updatedPresentation = newTweetsPill.presentation.map {

case presentation: UrtItemPresentation =>

presentation.timelineItem match {

case alert: ShowAlert =>

val text = if (useAvatars(query, userIds)) tweetedString else seeNewTweetsString

val richText = RichText(

text = stringCenter.prepare(text),

entities = List.empty,

rtl = None,

alignment = None)

val updatedAlert =

alert.copy(userIds = Some(userIds.take(NumAvatars)), richText = Some(richText))

presentation.copy(timelineItem = updatedAlert)

}

}

otherCandidates :+ newTweetsPill.copy(presentation = updatedPresentation)

}.getOrElse(remainingCandidates)

SelectorResult(remainingCandidates = updatedCandidates, result = result)

}

private def useAvatars(query: Query, userIds: Seq[Long]): Boolean = {

val enableAvatars = query.params(EnableNewTweetsPillAvatarsParam)

enableAvatars && userIds.size >= NumAvatars

}

}