package com.twitter.home\_mixer.functional\_component.side\_effect

import com.twitter.eventbus.client.EventBusPublisher

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

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

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

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

import com.twitter.home\_mixer.service.HomeMixerAlertConfig

import com.twitter.product\_mixer.core.functional\_component.side\_effect.PipelineResultSideEffect

import com.twitter.product\_mixer.core.model.common.identifier.SideEffectIdentifier

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

import com.twitter.product\_mixer.core.model.marshalling.HasMarshalling

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

import com.twitter.stitch.Stitch

import com.twitter.timelines.impressionstore.thriftscala.Impression

import com.twitter.timelines.impressionstore.thriftscala.ImpressionList

import com.twitter.timelines.impressionstore.thriftscala.PublishedImpressionList

import com.twitter.timelines.impressionstore.thriftscala.SurfaceArea

import com.twitter.util.Time

import javax.inject.Inject

import javax.inject.Singleton

object PublishClientSentImpressionsEventBusSideEffect {

val HomeSurfaceArea: Option[Set[SurfaceArea]] = Some(Set(SurfaceArea.HomeTimeline))

val HomeLatestSurfaceArea: Option[Set[SurfaceArea]] = Some(Set(SurfaceArea.HomeLatestTimeline))

val HomeSubscribedSurfaceArea: Option[Set[SurfaceArea]] = Some(Set(SurfaceArea.HomeSubscribed))

}

/\*\*

\* Side effect that publishes seen tweet IDs sent from clients. The seen tweet IDs are sent to a

\* heron topology which writes to a memcache dataset.

\*/

@Singleton

class PublishClientSentImpressionsEventBusSideEffect @Inject() (

eventBusPublisher: EventBusPublisher[PublishedImpressionList])

extends PipelineResultSideEffect[PipelineQuery with HasSeenTweetIds, HasMarshalling]

with PipelineResultSideEffect.Conditionally[

PipelineQuery with HasSeenTweetIds,

HasMarshalling

] {

import PublishClientSentImpressionsEventBusSideEffect.\_

override val identifier: SideEffectIdentifier =

SideEffectIdentifier("PublishClientSentImpressionsEventBus")

override def onlyIf(

query: PipelineQuery with HasSeenTweetIds,

selectedCandidates: Seq[CandidateWithDetails],

remainingCandidates: Seq[CandidateWithDetails],

droppedCandidates: Seq[CandidateWithDetails],

response: HasMarshalling

): Boolean = query.seenTweetIds.exists(\_.nonEmpty)

def buildEvents(

query: PipelineQuery with HasSeenTweetIds,

currentTime: Long

): Option[Seq[Impression]] = {

val surfaceArea = query.product match {

case ForYouProduct => HomeSurfaceArea

case FollowingProduct => HomeLatestSurfaceArea

case SubscribedProduct => HomeSubscribedSurfaceArea

case \_ => None

}

query.seenTweetIds.map { seenTweetIds =>

seenTweetIds.map { tweetId =>

Impression(

tweetId = tweetId,

impressionTime = Some(currentTime),

surfaceAreas = surfaceArea

)

}

}

}

final override def apply(

inputs: PipelineResultSideEffect.Inputs[PipelineQuery with HasSeenTweetIds, HasMarshalling]

): Stitch[Unit] = {

val currentTime = Time.now.inMilliseconds

val impressions = buildEvents(inputs.query, currentTime)

Stitch.callFuture(

eventBusPublisher.publish(

PublishedImpressionList(

inputs.query.getRequiredUserId,

ImpressionList(impressions),

currentTime

)

)

)

}

override val alerts = Seq(

HomeMixerAlertConfig.BusinessHours.defaultSuccessRateAlert(99.4)

)

}