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

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

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.impression.{thriftscala => t}

import com.twitter.timelines.impressionstore.store.ManhattanTweetImpressionStoreClient

import javax.inject.Inject

import javax.inject.Singleton

/\*\*

\* Side effect that updates the timelines tweet impression

\* store (Manhattan) with seen tweet IDs sent from clients

\*/

@Singleton

class PublishClientSentImpressionsManhattanSideEffect @Inject() (

manhattanTweetImpressionStoreClient: ManhattanTweetImpressionStoreClient)

extends PipelineResultSideEffect[PipelineQuery with HasSeenTweetIds, HasMarshalling]

with PipelineResultSideEffect.Conditionally[

PipelineQuery with HasSeenTweetIds,

HasMarshalling

] {

override val identifier: SideEffectIdentifier =

SideEffectIdentifier("PublishClientSentImpressionsManhattan")

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): Option[(Long, t.TweetImpressionsEntries)] = {

query.features.flatMap { featureMap =>

val impressions = featureMap.getOrElse(TweetImpressionsFeature, Seq.empty)

if (impressions.nonEmpty)

Some((query.getRequiredUserId, t.TweetImpressionsEntries(impressions)))

else None

}

}

final override def apply(

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

): Stitch[Unit] = {

val events = buildEvents(inputs.query)

Stitch

.traverse(events) {

case (key, value) => manhattanTweetImpressionStoreClient.write(key, value)

}

.unit

}

override val alerts = Seq(

HomeMixerAlertConfig.BusinessHours.defaultSuccessRateAlert(99.4)

)

}