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

import com.twitter.clientapp.thriftscala.LogEvent

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

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

import com.twitter.logpipeline.client.common.EventPublisher

import com.twitter.product\_mixer.component\_library.side\_effect.ScribeClientEventSideEffect

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

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

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.response.urt.Timeline

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

/\*\*

\* Side effect that logs served tweet metrics to Scribe as client events.

\*/

case class HomeScribeClientEventSideEffect(

enableScribeClientEvents: Boolean,

override val logPipelinePublisher: EventPublisher[LogEvent],

injectedTweetsCandidatePipelineIdentifiers: Seq[CandidatePipelineIdentifier],

adsCandidatePipelineIdentifier: Option[CandidatePipelineIdentifier] = None,

whoToFollowCandidatePipelineIdentifier: Option[CandidatePipelineIdentifier] = None,

whoToSubscribeCandidatePipelineIdentifier: Option[CandidatePipelineIdentifier] = None)

extends ScribeClientEventSideEffect[PipelineQuery, Timeline]

with PipelineResultSideEffect.Conditionally[

PipelineQuery,

Timeline

] {

override val identifier: SideEffectIdentifier = SideEffectIdentifier("HomeScribeClientEvent")

override val page = "timelinemixer"

override def onlyIf(

query: PipelineQuery,

selectedCandidates: Seq[CandidateWithDetails],

remainingCandidates: Seq[CandidateWithDetails],

droppedCandidates: Seq[CandidateWithDetails],

response: Timeline

): Boolean = enableScribeClientEvents

override def buildClientEvents(

query: PipelineQuery,

selectedCandidates: Seq[CandidateWithDetails],

remainingCandidates: Seq[CandidateWithDetails],

droppedCandidates: Seq[CandidateWithDetails],

response: Timeline

): Seq[ScribeClientEventSideEffect.ClientEvent] = {

val itemCandidates = CandidatesUtil.getItemCandidates(selectedCandidates)

val sources = itemCandidates.groupBy(\_.source)

val injectedTweets =

injectedTweetsCandidatePipelineIdentifiers.flatMap(sources.getOrElse(\_, Seq.empty))

val promotedTweets = adsCandidatePipelineIdentifier.flatMap(sources.get).toSeq.flatten

// WhoToFollow and WhoToSubscribe modules are not required for all home-mixer products, e.g. list tweets timeline.

val whoToFollowUsers = whoToFollowCandidatePipelineIdentifier.flatMap(sources.get).toSeq.flatten

val whoToSubscribeUsers =

whoToSubscribeCandidatePipelineIdentifier.flatMap(sources.get).toSeq.flatten

val servedEvents = ServedEventsBuilder

.build(query, injectedTweets, promotedTweets, whoToFollowUsers, whoToSubscribeUsers)

val emptyTimelineEvents = EmptyTimelineEventsBuilder.build(query, injectedTweets)

val queryEvents = QueryEventsBuilder.build(query, injectedTweets)

(servedEvents ++ emptyTimelineEvents ++ queryEvents).filter(\_.eventValue.forall(\_ > 0))

}

override val alerts = Seq(

HomeMixerAlertConfig.BusinessHours.defaultSuccessRateAlert(99.9)

)

}