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

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

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

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

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

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.marshalling.response.urt.Timeline

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

import com.twitter.stitch.Stitch

import com.twitter.timelinemixer.clients.persistence.TimelineResponseBatchesClient

import com.twitter.timelinemixer.clients.persistence.TimelineResponseV3

import com.twitter.timelineservice.model.TimelineQuery

import com.twitter.timelineservice.model.core.TimelineKind

import javax.inject.Inject

import javax.inject.Singleton

/\*\*

\* Side effect that truncates entries in the Timelines Persistence store

\* based on the number of entries per client.

\*/

@Singleton

class TruncateTimelinesPersistenceStoreSideEffect @Inject() (

timelineResponseBatchesClient: TimelineResponseBatchesClient[TimelineResponseV3])

extends PipelineResultSideEffect[PipelineQuery, Timeline] {

override val identifier: SideEffectIdentifier =

SideEffectIdentifier("TruncateTimelinesPersistenceStore")

def getResponsesToDelete(query: PipelineQuery): Seq[TimelineResponseV3] = {

val responses =

query.features.map(\_.getOrElse(PersistenceEntriesFeature, Seq.empty)).toSeq.flatten

val responsesByClient = responses.groupBy(\_.clientPlatform).values.toSeq

val maxEntriesPerClient = query.params(TimelinesPersistenceStoreMaxEntriesPerClient)

responsesByClient.flatMap {

\_.sortBy(\_.servedTime.inMilliseconds)

.foldRight((Seq.empty[TimelineResponseV3], maxEntriesPerClient)) {

case (response, (responsesToDelete, remainingCap)) =>

if (remainingCap > 0) (responsesToDelete, remainingCap - response.entries.size)

else (response +: responsesToDelete, remainingCap)

} match { case (responsesToDelete, \_) => responsesToDelete }

}

}

final override def apply(

inputs: PipelineResultSideEffect.Inputs[PipelineQuery, Timeline]

): Stitch[Unit] = {

val timelineKind = inputs.query.product match {

case FollowingProduct => TimelineKind.homeLatest

case ForYouProduct => TimelineKind.home

case other => throw new UnsupportedOperationException(s"Unknown product: $other")

}

val timelineQuery = TimelineQuery(id = inputs.query.getRequiredUserId, kind = timelineKind)

val responsesToDelete = getResponsesToDelete(inputs.query)

if (responsesToDelete.nonEmpty)

Stitch.callFuture(timelineResponseBatchesClient.delete(timelineQuery, responsesToDelete))

else Stitch.Unit

}

override val alerts = Seq(

HomeMixerAlertConfig.BusinessHours.defaultSuccessRateAlert(99.8)

)

}