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

import com.twitter.common\_internal.analytics.twitter\_client\_user\_agent\_parser.UserAgent

import com.twitter.product\_mixer.core.feature.Feature

import com.twitter.product\_mixer.core.functional\_component.gate.Gate

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

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

import com.twitter.stitch.Stitch

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

import com.twitter.timelinemixer.injection.store.persistence.TimelinePersistenceUtils

import com.twitter.timelines.configapi.Param

import com.twitter.timelines.util.client\_info.ClientPlatform

import com.twitter.timelineservice.model.rich.EntityIdType

import com.twitter.util.Duration

import com.twitter.util.Time

/\*\*

\* Gate used to reduce the frequency of injections. Note that the actual interval between injections may be

\* less than the specified minInjectionIntervalParam if data is unavailable or missing. For example, being deleted by

\* the persistence store via a TTL or similar mechanism.

\*

\* @param minInjectionIntervalParam the desired minimum interval between injections

\* @param persistenceEntriesFeature the feature for retrieving persisted timeline responses

\*/

case class TimelinesPersistenceStoreLastInjectionGate(

minInjectionIntervalParam: Param[Duration],

persistenceEntriesFeature: Feature[PipelineQuery, Seq[TimelineResponseV3]],

entityIdType: EntityIdType.Value)

extends Gate[PipelineQuery]

with TimelinePersistenceUtils {

override val identifier: GateIdentifier = GateIdentifier("TimelinesPersistenceStoreLastInjection")

override def shouldContinue(query: PipelineQuery): Stitch[Boolean] =

Stitch(

query.queryTime.since(getLastInjectionTime(query)) > query.params(minInjectionIntervalParam))

private def getLastInjectionTime(query: PipelineQuery) = query.features

.flatMap { featureMap =>

val timelineResponses = featureMap.getOrElse(persistenceEntriesFeature, Seq.empty)

val clientPlatform = ClientPlatform.fromQueryOptions(

clientAppId = query.clientContext.appId,

userAgent = query.clientContext.userAgent.flatMap(UserAgent.fromString)

)

val sortedResponses = responseByClient(clientPlatform, timelineResponses)

val latestResponseWithEntityIdTypeEntry =

sortedResponses.find(\_.entries.exists(\_.entityIdType == entityIdType))

latestResponseWithEntityIdTypeEntry.map(\_.servedTime)

}.getOrElse(Time.Bottom)

}