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

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

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

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

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

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

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

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

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

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

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

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

import com.twitter.timelineservice.model.util.FinagleRequestContext

import com.twitter.user\_session\_store.ReadWriteUserSessionStore

import com.twitter.user\_session\_store.WriteRequest

import com.twitter.user\_session\_store.thriftscala.NonPollingTimestamps

import com.twitter.user\_session\_store.thriftscala.UserSessionField

import com.twitter.util.Time

import javax.inject.Inject

import javax.inject.Singleton

/\*\*

\* Side effect that updates the User Session Store (Manhattan) with the timestamps of non polling requests.

\*/

@Singleton

class UpdateLastNonPollingTimeSideEffect[

Query <: PipelineQuery with HasDeviceContext,

ResponseType <: HasMarshalling] @Inject() (

override val userSessionStore: ReadWriteUserSessionStore)

extends UserSessionStoreUpdateSideEffect[

WriteRequest,

Query,

ResponseType

] {

private val MaxNonPollingTimes = 10

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

/\*\*

\* When the request is non polling and is not a background fetch request, update

\* the list of non polling timestamps with the timestamp of the current request

\*/

override def buildWriteRequest(query: Query): Option[WriteRequest] = {

val isBackgroundFetch = query.deviceContext

.exists(\_.requestContextValue.contains(DeviceContext.RequestContext.BackgroundFetch))

if (!query.features.exists(\_.getOrElse(PollingFeature, false)) && !isBackgroundFetch) {

val fields = Seq(UserSessionField.NonPollingTimestamps(makeLastNonPollingTimestamps(query)))

Some(WriteRequest(query.getRequiredUserId, fields))

} else None

}

override val alerts = Seq(

HomeMixerAlertConfig.BusinessHours.defaultSuccessRateAlert(99.96)

)

private def makeLastNonPollingTimestamps(query: Query): NonPollingTimestamps = {

val priorNonPollingTimestamps =

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

val lastNonPollingTimeMs =

FinagleRequestContext.default.requestStartTime.get.getOrElse(Time.now).inMillis

val followingLastNonPollingTime = query.features

.flatMap(features => features.getOrElse(FollowingLastNonPollingTimeFeature, None))

.map(\_.inMillis)

NonPollingTimestamps(

nonPollingTimestampsMs =

(lastNonPollingTimeMs +: priorNonPollingTimestamps).take(MaxNonPollingTimes),

mostRecentHomeLatestNonPollingTimestampMs =

if (query.product == FollowingProduct) Some(lastNonPollingTimeMs)

else followingLastNonPollingTime

)

}

}