package com.twitter.home\_mixer.product.list\_tweets

import com.twitter.conversions.DurationOps.\_

import com.twitter.home\_mixer.marshaller.timelines.ChronologicalCursorUnmarshaller

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

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

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

import com.twitter.home\_mixer.product.list\_tweets.model.ListTweetsQuery

import com.twitter.home\_mixer.product.list\_tweets.param.ListTweetsParam.ServerMaxResultsParam

import com.twitter.home\_mixer.product.list\_tweets.param.ListTweetsParamConfig

import com.twitter.home\_mixer.service.HomeMixerAccessPolicy.DefaultHomeMixerAccessPolicy

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

import com.twitter.product\_mixer.component\_library.model.cursor.UrtOrderedCursor

import com.twitter.product\_mixer.component\_library.premarshaller.cursor.UrtCursorSerializer

import com.twitter.product\_mixer.core.functional\_component.common.access\_policy.AccessPolicy

import com.twitter.product\_mixer.core.functional\_component.common.alert.Alert

import com.twitter.product\_mixer.core.functional\_component.common.alert.EmptyResponseRateAlert

import com.twitter.product\_mixer.core.functional\_component.common.alert.LatencyAlert

import com.twitter.product\_mixer.core.functional\_component.common.alert.P99

import com.twitter.product\_mixer.core.functional\_component.common.alert.SuccessRateAlert

import com.twitter.product\_mixer.core.functional\_component.common.alert.ThroughputAlert

import com.twitter.product\_mixer.core.functional\_component.common.alert.predicate.TriggerIfAbove

import com.twitter.product\_mixer.core.functional\_component.common.alert.predicate.TriggerIfBelow

import com.twitter.product\_mixer.core.functional\_component.common.alert.predicate.TriggerIfLatencyAbove

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

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

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

import com.twitter.product\_mixer.core.model.marshalling.response.urt.operation.GapCursor

import com.twitter.product\_mixer.core.model.marshalling.response.urt.operation.TopCursor

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

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.BadRequest

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.MalformedCursor

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.PipelineFailure

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

import com.twitter.product\_mixer.core.product.ProductParamConfig

import com.twitter.product\_mixer.core.util.SortIndexBuilder

import com.twitter.timelines.configapi.Params

import com.twitter.timelines.render.{thriftscala => urt}

import com.twitter.timelines.util.RequestCursorSerializer

import com.twitter.util.Time

import com.twitter.util.Try

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class ListTweetsProductPipelineConfig @Inject() (

listTweetsMixerPipelineConfig: ListTweetsMixerPipelineConfig,

listTweetsParamConfig: ListTweetsParamConfig)

extends ProductPipelineConfig[HomeMixerRequest, ListTweetsQuery, urt.TimelineResponse] {

override val identifier: ProductPipelineIdentifier = ProductPipelineIdentifier("ListTweets")

override val product: request.Product = ListTweetsProduct

override val paramConfig: ProductParamConfig = listTweetsParamConfig

override val denyLoggedOutUsers: Boolean = false

override def pipelineQueryTransformer(

request: HomeMixerRequest,

params: Params

): ListTweetsQuery = {

val context = request.productContext match {

case Some(context: ListTweetsProductContext) => context

case \_ => throw PipelineFailure(BadRequest, "ListTweetsProductContext not found")

}

val debugOptions = request.debugParams.flatMap(\_.debugOptions)

/\*\*

\* Unlike other clients, newly created tweets on Android have the sort index set to the current

\* time instead of the top sort index + 1, so these tweets get stuck at the top of the timeline

\* if subsequent timeline responses use the sort index from the previous response instead of

\* the current time.

\*/

val pipelineCursor = request.serializedRequestCursor.flatMap { cursor =>

Try(UrtCursorSerializer.deserializeOrderedCursor(cursor))

.getOrElse(ChronologicalCursorUnmarshaller(RequestCursorSerializer.deserialize(cursor)))

.map {

case UrtOrderedCursor(\_, id, Some(GapCursor), gapBoundaryId)

if id.isEmpty || gapBoundaryId.isEmpty =>

throw PipelineFailure(MalformedCursor, "Gap Cursor bounds not defined")

case topCursor @ UrtOrderedCursor(\_, \_, Some(TopCursor), \_) =>

val queryTime = debugOptions.flatMap(\_.requestTimeOverride).getOrElse(Time.now)

topCursor.copy(initialSortIndex = SortIndexBuilder.timeToId(queryTime))

case cursor => cursor

}

}

ListTweetsQuery(

params = params,

clientContext = request.clientContext,

features = None,

pipelineCursor = pipelineCursor,

requestedMaxResults = Some(params(ServerMaxResultsParam)),

debugOptions = debugOptions,

listId = context.listId,

deviceContext = context.deviceContext,

dspClientContext = context.dspClientContext

)

}

override def pipelines: Seq[PipelineConfig] = Seq(listTweetsMixerPipelineConfig)

override def pipelineSelector(query: ListTweetsQuery): ComponentIdentifier =

listTweetsMixerPipelineConfig.identifier

override val alerts: Seq[Alert] = Seq(

SuccessRateAlert(

notificationGroup = DefaultNotificationGroup,

warnPredicate = TriggerIfBelow(99.9, 20, 30),

criticalPredicate = TriggerIfBelow(99.9, 30, 30),

),

LatencyAlert(

notificationGroup = DefaultNotificationGroup,

percentile = P99,

warnPredicate = TriggerIfLatencyAbove(300.millis, 15, 30),

criticalPredicate = TriggerIfLatencyAbove(400.millis, 15, 30)

),

ThroughputAlert(

notificationGroup = DefaultNotificationGroup,

warnPredicate = TriggerIfAbove(3000),

criticalPredicate = TriggerIfAbove(4000)

),

EmptyResponseRateAlert(

notificationGroup = DefaultNotificationGroup,

warnPredicate = TriggerIfAbove(65),

criticalPredicate = TriggerIfAbove(80)

)

)

override val debugAccessPolicies: Set[AccessPolicy] = DefaultHomeMixerAccessPolicy

}