package com.twitter.home\_mixer.product.scored\_tweets.side\_effect

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

import com.twitter.home\_mixer.product.scored\_tweets.model.ScoredTweetsResponse

import com.twitter.home\_mixer.product.scored\_tweets.param.ScoredTweetsParam.CachedScoredTweets

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

import com.twitter.home\_mixer.{thriftscala => hmt}

import com.twitter.product\_mixer.core.functional\_component.marshaller.response.urt.metadata.TopicContextFunctionalityTypeMarshaller

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.common.presentation.CandidateWithDetails

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

import com.twitter.servo.cache.TtlCache

import com.twitter.stitch.Stitch

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class CachedScoredTweetsSideEffect @Inject() (

scoredTweetsCache: TtlCache[Long, hmt.ScoredTweetsResponse])

extends PipelineResultSideEffect[PipelineQuery, ScoredTweetsResponse] {

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

private val MaxTweetsToCache = 1000

def buildCachedScoredTweets(

query: PipelineQuery,

candidates: Seq[CandidateWithDetails]

): hmt.ScoredTweetsResponse = {

val tweets = candidates.map { candidate =>

val sgsValidLikedByUserIds =

candidate.features.getOrElse(SGSValidLikedByUserIdsFeature, Seq.empty)

val sgsValidFollowedByUserIds =

candidate.features.getOrElse(SGSValidFollowedByUserIdsFeature, Seq.empty)

val perspectiveFilteredLikedByUserIds =

candidate.features.getOrElse(PerspectiveFilteredLikedByUserIdsFeature, Seq.empty)

val ancestors = candidate.features.getOrElse(AncestorsFeature, Seq.empty)

hmt.ScoredTweet(

tweetId = candidate.candidateIdLong,

authorId = candidate.features.get(AuthorIdFeature).get,

// Cache the model score instead of the final score because rescoring is per-request

score = candidate.features.getOrElse(WeightedModelScoreFeature, None),

suggestType = candidate.features.getOrElse(SuggestTypeFeature, None),

sourceTweetId = candidate.features.getOrElse(SourceTweetIdFeature, None),

sourceUserId = candidate.features.getOrElse(SourceUserIdFeature, None),

quotedTweetId = candidate.features.getOrElse(QuotedTweetIdFeature, None),

quotedUserId = candidate.features.getOrElse(QuotedUserIdFeature, None),

inReplyToTweetId = candidate.features.getOrElse(InReplyToTweetIdFeature, None),

inReplyToUserId = candidate.features.getOrElse(InReplyToUserIdFeature, None),

directedAtUserId = candidate.features.getOrElse(DirectedAtUserIdFeature, None),

inNetwork = Some(candidate.features.getOrElse(InNetworkFeature, true)),

sgsValidLikedByUserIds = Some(sgsValidLikedByUserIds),

sgsValidFollowedByUserIds = Some(sgsValidFollowedByUserIds),

topicId = candidate.features.getOrElse(TopicIdSocialContextFeature, None),

topicFunctionalityType = candidate.features

.getOrElse(TopicContextFunctionalityTypeFeature, None).map(

TopicContextFunctionalityTypeMarshaller(\_)),

ancestors = if (ancestors.nonEmpty) Some(ancestors) else None,

isReadFromCache = Some(true),

streamToKafka = Some(false),

exclusiveConversationAuthorId = candidate.features

.getOrElse(ExclusiveConversationAuthorIdFeature, None),

authorMetadata = Some(

hmt.AuthorMetadata(

blueVerified = candidate.features.getOrElse(AuthorIsBlueVerifiedFeature, false),

goldVerified = candidate.features.getOrElse(AuthorIsGoldVerifiedFeature, false),

grayVerified = candidate.features.getOrElse(AuthorIsGrayVerifiedFeature, false),

legacyVerified = candidate.features.getOrElse(AuthorIsLegacyVerifiedFeature, false),

creator = candidate.features.getOrElse(AuthorIsCreatorFeature, false)

)),

lastScoredTimestampMs = candidate.features

.getOrElse(LastScoredTimestampMsFeature, Some(query.queryTime.inMilliseconds)),

candidatePipelineIdentifier = candidate.features

.getOrElse(CachedCandidatePipelineIdentifierFeature, Some(candidate.source.name)),

tweetUrls = Some(candidate.features.getOrElse(TweetUrlsFeature, Seq.empty)),

perspectiveFilteredLikedByUserIds = Some(perspectiveFilteredLikedByUserIds)

)

}

hmt.ScoredTweetsResponse(tweets)

}

final override def apply(

inputs: PipelineResultSideEffect.Inputs[PipelineQuery, ScoredTweetsResponse]

): Stitch[Unit] = {

val candidates =

(inputs.selectedCandidates ++ inputs.remainingCandidates ++ inputs.droppedCandidates)

.filter(\_.features.getOrElse(ScoreFeature, None).exists(\_ > 0.0))

val truncatedCandidates =

if (candidates.size > MaxTweetsToCache)

candidates

.sortBy(-\_.features.getOrElse(ScoreFeature, None).getOrElse(0.0)).take(MaxTweetsToCache)

else candidates

if (truncatedCandidates.nonEmpty) {

val ttl = inputs.query.params(CachedScoredTweets.TTLParam)

val scoredTweets = buildCachedScoredTweets(inputs.query, truncatedCandidates)

Stitch.callFuture(scoredTweetsCache.set(inputs.query.getRequiredUserId, scoredTweets, ttl))

} else Stitch.Unit

}

override val alerts = Seq(

HomeMixerAlertConfig.BusinessHours.defaultSuccessRateAlert(99.4)

)

}