package com.twitter.home\_mixer.product.for\_you.candidate\_source

import com.google.inject.Provider

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

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

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

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

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

import com.twitter.home\_mixer.product.for\_you.model.ForYouQuery

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

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

import com.twitter.product\_mixer.core.functional\_component.candidate\_source.product\_pipeline.ProductPipelineCandidateSource

import com.twitter.product\_mixer.core.functional\_component.configapi.ParamsBuilder

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

import com.twitter.product\_mixer.core.product.registry.ProductPipelineRegistry

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

import com.twitter.timelineservice.suggests.{thriftscala => st}

import com.twitter.tweetconvosvc.tweet\_ancestor.{thriftscala => ta}

import javax.inject.Inject

import javax.inject.Singleton

/\*\*

\* [[ScoredTweetWithConversationMetadata]]

\*\*/

case class ScoredTweetWithConversationMetadata(

tweetId: Long,

authorId: Long,

score: Option[Double] = None,

suggestType: Option[st.SuggestType] = None,

sourceTweetId: Option[Long] = None,

sourceUserId: Option[Long] = None,

quotedTweetId: Option[Long] = None,

quotedUserId: Option[Long] = None,

inReplyToTweetId: Option[Long] = None,

inReplyToUserId: Option[Long] = None,

directedAtUserId: Option[Long] = None,

inNetwork: Option[Boolean] = None,

sgsValidLikedByUserIds: Option[Seq[Long]] = None,

sgsValidFollowedByUserIds: Option[Seq[Long]] = None,

ancestors: Option[Seq[ta.TweetAncestor]] = None,

topicId: Option[Long] = None,

topicFunctionalityType: Option[tl.TopicContextFunctionalityType] = None,

conversationId: Option[Long] = None,

conversationFocalTweetId: Option[Long] = None,

isReadFromCache: Option[Boolean] = None,

streamToKafka: Option[Boolean] = None,

exclusiveConversationAuthorId: Option[Long] = None,

authorIsBlueVerified: Option[Boolean] = None,

authorIsGoldVerified: Option[Boolean] = None,

authorIsGrayVerified: Option[Boolean] = None,

authorIsLegacyVerified: Option[Boolean] = None,

authorIsCreator: Option[Boolean] = None,

perspectiveFilteredLikedByUserIds: Option[Seq[Long]] = None)

@Singleton

class ScoredTweetsProductCandidateSource @Inject() (

override val productPipelineRegistry: Provider[ProductPipelineRegistry],

override val paramsBuilder: Provider[ParamsBuilder])

extends ProductPipelineCandidateSource[

ForYouQuery,

HomeMixerRequest,

t.ScoredTweetsResponse,

ScoredTweetWithConversationMetadata

] {

override val identifier: CandidateSourceIdentifier =

CandidateSourceIdentifier("ScoredTweetsProduct")

private val MaxModuleSize = 3

private val MaxAncestorsInConversation = 2

override def pipelineRequestTransformer(productPipelineQuery: ForYouQuery): HomeMixerRequest = {

HomeMixerRequest(

clientContext = productPipelineQuery.clientContext,

product = ScoredTweetsProduct,

productContext = Some(

ScoredTweetsProductContext(

productPipelineQuery.deviceContext,

productPipelineQuery.seenTweetIds,

productPipelineQuery.features.map(\_.getOrElse(ServedTweetIdsFeature, Seq.empty)),

productPipelineQuery.features.map(\_.getOrElse(TimelineServiceTweetsFeature, Seq.empty))

)),

serializedRequestCursor =

productPipelineQuery.pipelineCursor.map(UrtCursorSerializer.serializeCursor),

maxResults = productPipelineQuery.requestedMaxResults,

debugParams = None,

homeRequestParam = false

)

}

override def productPipelineResultTransformer(

productPipelineResult: t.ScoredTweetsResponse

): Seq[ScoredTweetWithConversationMetadata] = {

val scoredTweets = productPipelineResult.scoredTweets.flatMap { focalTweet =>

val parentTweets = focalTweet.ancestors.getOrElse(Seq.empty).sortBy(-\_.tweetId)

val (intermediates, root) = parentTweets.splitAt(parentTweets.size - 1)

val truncatedIntermediates =

intermediates.take(MaxModuleSize - MaxAncestorsInConversation).reverse

val rootScoredTweet: Seq[ScoredTweetWithConversationMetadata] = root.map { ancestor =>

ScoredTweetWithConversationMetadata(

tweetId = ancestor.tweetId,

authorId = ancestor.userId,

suggestType = focalTweet.suggestType,

conversationId = Some(ancestor.tweetId),

conversationFocalTweetId = Some(focalTweet.tweetId),

exclusiveConversationAuthorId = focalTweet.exclusiveConversationAuthorId

)

}

val conversationId = rootScoredTweet.headOption.map(\_.tweetId)

val tweetsToParents =

if (parentTweets.nonEmpty) parentTweets.zip(parentTweets.tail).toMap

else Map.empty[ta.TweetAncestor, ta.TweetAncestor]

val intermediateScoredTweets = truncatedIntermediates.map { ancestor =>

ScoredTweetWithConversationMetadata(

tweetId = ancestor.tweetId,

authorId = ancestor.userId,

suggestType = focalTweet.suggestType,

inReplyToTweetId = tweetsToParents.get(ancestor).map(\_.tweetId),

conversationId = conversationId,

conversationFocalTweetId = Some(focalTweet.tweetId),

exclusiveConversationAuthorId = focalTweet.exclusiveConversationAuthorId

)

}

val parentScoredTweets = rootScoredTweet ++ intermediateScoredTweets

val conversationFocalTweetId =

if (parentScoredTweets.nonEmpty) Some(focalTweet.tweetId) else None

val focalScoredTweet = ScoredTweetWithConversationMetadata(

tweetId = focalTweet.tweetId,

authorId = focalTweet.authorId,

score = focalTweet.score,

suggestType = focalTweet.suggestType,

sourceTweetId = focalTweet.sourceTweetId,

sourceUserId = focalTweet.sourceUserId,

quotedTweetId = focalTweet.quotedTweetId,

quotedUserId = focalTweet.quotedUserId,

inReplyToTweetId = parentScoredTweets.lastOption.map(\_.tweetId),

inReplyToUserId = focalTweet.inReplyToUserId,

directedAtUserId = focalTweet.directedAtUserId,

inNetwork = focalTweet.inNetwork,

sgsValidLikedByUserIds = focalTweet.sgsValidLikedByUserIds,

sgsValidFollowedByUserIds = focalTweet.sgsValidFollowedByUserIds,

topicId = focalTweet.topicId,

topicFunctionalityType = focalTweet.topicFunctionalityType,

ancestors = focalTweet.ancestors,

conversationId = conversationId,

conversationFocalTweetId = conversationFocalTweetId,

isReadFromCache = focalTweet.isReadFromCache,

streamToKafka = focalTweet.streamToKafka,

exclusiveConversationAuthorId = focalTweet.exclusiveConversationAuthorId,

authorIsBlueVerified = focalTweet.authorMetadata.map(\_.blueVerified),

authorIsGoldVerified = focalTweet.authorMetadata.map(\_.goldVerified),

authorIsGrayVerified = focalTweet.authorMetadata.map(\_.grayVerified),

authorIsLegacyVerified = focalTweet.authorMetadata.map(\_.legacyVerified),

authorIsCreator = focalTweet.authorMetadata.map(\_.creator),

perspectiveFilteredLikedByUserIds = focalTweet.perspectiveFilteredLikedByUserIds

)

parentScoredTweets :+ focalScoredTweet

}

val dedupedTweets = scoredTweets.groupBy(\_.tweetId).map {

case (\_, duplicateAncestors) => duplicateAncestors.maxBy(\_.score.getOrElse(0.0))

}

// Sort by tweet id to prevent issues with future assumptions of the root being the first

// tweet and the focal being the last tweet in a module. The tweets as a whole do not need

// to be sorted overall, only the relative order within modules must be kept.

dedupedTweets.toSeq.sortBy(\_.tweetId)

}

}