package com.twitter.home\_mixer.functional\_component.feature\_hydrator

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

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

import com.twitter.product\_mixer.component\_library.model.candidate.TweetCandidate

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

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMapBuilder

import com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.BulkCandidateFeatureHydrator

import com.twitter.product\_mixer.core.model.common.CandidateWithFeatures

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

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

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

import com.twitter.stitch.Stitch

import com.twitter.stitch.timelineservice.TimelineService

import com.twitter.stitch.timelineservice.TimelineService.GetPerspectives

import com.twitter.timelineservice.thriftscala.PerspectiveType

import com.twitter.timelineservice.thriftscala.PerspectiveType.Favorited

import javax.inject.Inject

import javax.inject.Singleton

/\*\*

\* Filter out unlike edges from liked-by tweets

\* Useful if the likes come from a cache and because UTEG does not fully remove unlike edges.

\*/

@Singleton

class PerspectiveFilteredSocialContextFeatureHydrator @Inject() (timelineService: TimelineService)

extends BulkCandidateFeatureHydrator[PipelineQuery, TweetCandidate] {

override val identifier: FeatureHydratorIdentifier =

FeatureHydratorIdentifier("PerspectiveFilteredSocialContext")

override val features: Set[Feature[\_, \_]] = Set(PerspectiveFilteredLikedByUserIdsFeature)

private val MaxCountUsers = 10

private val favoritePerspectiveSet: Set[PerspectiveType] = Set(Favorited)

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[Seq[FeatureMap]] = OffloadFuturePools.offloadStitch {

val engagingUserIdtoTweetId = candidates.flatMap { candidate =>

candidate.features

.getOrElse(FavoritedByUserIdsFeature, Seq.empty).take(MaxCountUsers)

.map(favoritedBy => favoritedBy -> candidate.candidate.id)

}

val queries = engagingUserIdtoTweetId.map {

case (userId, tweetId) =>

GetPerspectives.Query(userId = userId, tweetId = tweetId, types = favoritePerspectiveSet)

}

Stitch.collect(queries.map(timelineService.getPerspective)).map { perspectiveResults =>

val validUserIdTweetIds: Set[(Long, Long)] =

queries

.zip(perspectiveResults)

.collect { case (query, perspective) if perspective.favorited => query }

.map(query => (query.userId, query.tweetId))

.toSet

candidates.map { candidate =>

val perspectiveFilteredFavoritedByUserIds: Seq[Long] = candidate.features

.getOrElse(FavoritedByUserIdsFeature, Seq.empty).take(MaxCountUsers)

.filter { userId => validUserIdTweetIds.contains((userId, candidate.candidate.id)) }

FeatureMapBuilder()

.add(PerspectiveFilteredLikedByUserIdsFeature, perspectiveFilteredFavoritedByUserIds)

.build()

}

}

}

}