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

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

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

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

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

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.socialgraph.{thriftscala => sg}

import com.twitter.stitch.Stitch

import com.twitter.stitch.socialgraph.SocialGraph

import javax.inject.Inject

import javax.inject.Singleton

/\*\*

\* This hydrator takes liked-by and followed-by user ids and checks via SGS that the viewer is

\* following the engager, that the viewer is not blocking the engager, that the engager is not

\* blocking the viewer, and that the viewer has not muted the engager.

\*/

@Singleton

class SGSValidSocialContextFeatureHydrator @Inject() (

socialGraph: SocialGraph)

extends BulkCandidateFeatureHydrator[PipelineQuery, TweetCandidate] {

override val identifier: FeatureHydratorIdentifier =

FeatureHydratorIdentifier("SGSValidSocialContext")

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

SGSValidFollowedByUserIdsFeature,

SGSValidLikedByUserIdsFeature

)

private val MaxCountUsers = 10

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

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

val allSocialContextUserIds =

candidates.flatMap { candidate =>

candidate.features.getOrElse(FavoritedByUserIdsFeature, Nil).take(MaxCountUsers) ++

candidate.features.getOrElse(FollowedByUserIdsFeature, Nil).take(MaxCountUsers)

}.distinct

getValidUserIds(query.getRequiredUserId, allSocialContextUserIds).map { validUserIds =>

candidates.map { candidate =>

val sgsFilteredLikedByUserIds =

candidate.features

.getOrElse(FavoritedByUserIdsFeature, Nil).take(MaxCountUsers)

.filter(validUserIds.contains)

val sgsFilteredFollowedByUserIds =

candidate.features

.getOrElse(FollowedByUserIdsFeature, Nil).take(MaxCountUsers)

.filter(validUserIds.contains)

FeatureMapBuilder()

.add(SGSValidFollowedByUserIdsFeature, sgsFilteredFollowedByUserIds)

.add(SGSValidLikedByUserIdsFeature, sgsFilteredLikedByUserIds)

.build()

}

}

}

private def getValidUserIds(

viewerId: Long,

socialProofUserIds: Seq[Long]

): Stitch[Seq[Long]] = {

if (socialProofUserIds.nonEmpty) {

val request = sg.IdsRequest(

relationships = Seq(

sg.SrcRelationship(

viewerId,

sg.RelationshipType.Following,

targets = Some(socialProofUserIds),

hasRelationship = true),

sg.SrcRelationship(

viewerId,

sg.RelationshipType.Blocking,

targets = Some(socialProofUserIds),

hasRelationship = false),

sg.SrcRelationship(

viewerId,

sg.RelationshipType.BlockedBy,

targets = Some(socialProofUserIds),

hasRelationship = false),

sg.SrcRelationship(

viewerId,

sg.RelationshipType.Muting,

targets = Some(socialProofUserIds),

hasRelationship = false)

),

pageRequest = Some(sg.PageRequest(selectAll = Some(true)))

)

socialGraph.ids(request).map(\_.ids)

} else Stitch.Nil

}

}