package com.twitter.visibility.builder.spaces

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.gizmoduck.thriftscala.Label

import com.twitter.gizmoduck.thriftscala.MuteSurface

import com.twitter.stitch.Stitch

import com.twitter.visibility.builder.FeatureMapBuilder

import com.twitter.visibility.builder.common.MutedKeywordFeatures

import com.twitter.visibility.builder.users.AuthorFeatures

import com.twitter.visibility.builder.users.RelationshipFeatures

import com.twitter.visibility.common.AudioSpaceSource

import com.twitter.visibility.common.SpaceId

import com.twitter.visibility.common.SpaceSafetyLabelMapSource

import com.twitter.visibility.common.UserId

import com.twitter.visibility.features.\_

import com.twitter.visibility.models.{MutedKeyword => VfMutedKeyword}

import com.twitter.visibility.models.SafetyLabel

import com.twitter.visibility.models.SpaceSafetyLabel

import com.twitter.visibility.models.SpaceSafetyLabelType

class SpaceFeatures(

spaceSafetyLabelMap: StratoSpaceLabelMaps,

authorFeatures: AuthorFeatures,

relationshipFeatures: RelationshipFeatures,

mutedKeywordFeatures: MutedKeywordFeatures,

audioSpaceSource: AudioSpaceSource) {

def forSpaceAndAuthorIds(

spaceId: SpaceId,

viewerId: Option[UserId],

authorIds: Option[Seq[UserId]]

): FeatureMapBuilder => FeatureMapBuilder = {

\_.withFeature(SpaceSafetyLabels, spaceSafetyLabelMap.forSpaceId(spaceId))

.withFeature(AuthorId, getSpaceAuthors(spaceId, authorIds).map(\_.toSet))

.withFeature(AuthorUserLabels, allSpaceAuthorLabels(spaceId, authorIds))

.withFeature(ViewerFollowsAuthor, viewerFollowsAnySpaceAuthor(spaceId, authorIds, viewerId))

.withFeature(ViewerMutesAuthor, viewerMutesAnySpaceAuthor(spaceId, authorIds, viewerId))

.withFeature(ViewerBlocksAuthor, viewerBlocksAnySpaceAuthor(spaceId, authorIds, viewerId))

.withFeature(AuthorBlocksViewer, anySpaceAuthorBlocksViewer(spaceId, authorIds, viewerId))

.withFeature(

ViewerMutesKeywordInSpaceTitleForNotifications,

titleContainsMutedKeyword(

audioSpaceSource.getSpaceTitle(spaceId),

audioSpaceSource.getSpaceLanguage(spaceId),

viewerId)

)

}

def titleContainsMutedKeyword(

titleOptStitch: Stitch[Option[String]],

languageOptStitch: Stitch[Option[String]],

viewerId: Option[UserId],

): Stitch[VfMutedKeyword] = {

titleOptStitch.flatMap {

case None => Stitch.value(VfMutedKeyword(None))

case Some(spaceTitle) =>

languageOptStitch.flatMap { languageOpt =>

mutedKeywordFeatures.spaceTitleContainsMutedKeyword(

spaceTitle,

languageOpt,

mutedKeywordFeatures.allMutedKeywords(viewerId),

MuteSurface.Notifications)

}

}

}

def getSpaceAuthors(

spaceId: SpaceId,

authorIdsFromRequest: Option[Seq[UserId]]

): Stitch[Seq[UserId]] = {

authorIdsFromRequest match {

case Some(authorIds) => Stitch.apply(authorIds)

case \_ => audioSpaceSource.getAdminIds(spaceId)

}

}

def allSpaceAuthorLabels(

spaceId: SpaceId,

authorIdsFromRequest: Option[Seq[UserId]]

): Stitch[Seq[Label]] = {

getSpaceAuthors(spaceId, authorIdsFromRequest)

.flatMap(authorIds =>

Stitch.collect(authorIds.map(authorId => authorFeatures.authorUserLabels(authorId)))).map(

\_.flatten)

}

def viewerMutesAnySpaceAuthor(

spaceId: SpaceId,

authorIdsFromRequest: Option[Seq[UserId]],

viewerId: Option[UserId]

): Stitch[Boolean] = {

getSpaceAuthors(spaceId, authorIdsFromRequest)

.flatMap(authorIds =>

Stitch.collect(authorIds.map(authorId =>

relationshipFeatures.viewerMutesAuthor(authorId, viewerId)))).map(\_.contains(true))

}

def anySpaceAuthorBlocksViewer(

spaceId: SpaceId,

authorIdsFromRequest: Option[Seq[UserId]],

viewerId: Option[UserId]

): Stitch[Boolean] = {

getSpaceAuthors(spaceId, authorIdsFromRequest)

.flatMap(authorIds =>

Stitch.collect(authorIds.map(authorId =>

relationshipFeatures.authorBlocksViewer(authorId, viewerId)))).map(\_.contains(true))

}

}

class StratoSpaceLabelMaps(

spaceSafetyLabelSource: SpaceSafetyLabelMapSource,

statsReceiver: StatsReceiver) {

private[this] val scopedStatsReceiver = statsReceiver.scope("space\_features")

private[this] val spaceSafetyLabelsStats =

scopedStatsReceiver.scope(SpaceSafetyLabels.name).counter("requests")

def forSpaceId(

spaceId: SpaceId,

): Stitch[Seq[SpaceSafetyLabel]] = {

spaceSafetyLabelSource

.fetch(spaceId).map(\_.flatMap(\_.labels.map { stratoSafetyLabelMap =>

stratoSafetyLabelMap

.map(label =>

SpaceSafetyLabel(

SpaceSafetyLabelType.fromThrift(label.\_1),

SafetyLabel.fromThrift(label.\_2)))

}).toSeq.flatten).ensure(spaceSafetyLabelsStats.incr)

}

}