package com.twitter.visibility.builder.dms

import com.twitter.convosvc.thriftscala.ConversationQuery

import com.twitter.convosvc.thriftscala.ConversationQueryOptions

import com.twitter.convosvc.thriftscala.ConversationType

import com.twitter.convosvc.thriftscala.TimelineLookupState

import com.twitter.stitch.NotFound

import com.twitter.stitch.Stitch

import com.twitter.visibility.builder.FeatureMapBuilder

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

import com.twitter.visibility.common.DmConversationId

import com.twitter.visibility.common.UserId

import com.twitter.visibility.common.dm\_sources.DmConversationSource

import com.twitter.visibility.features.\_

case class InvalidDmConversationFeatureException(message: String) extends Exception(message)

class DmConversationFeatures(

dmConversationSource: DmConversationSource,

authorFeatures: AuthorFeatures) {

def forDmConversationId(

dmConversationId: DmConversationId,

viewerIdOpt: Option[UserId]

): FeatureMapBuilder => FeatureMapBuilder =

\_.withFeature(

DmConversationIsOneToOneConversation,

dmConversationIsOneToOneConversation(dmConversationId, viewerIdOpt))

.withFeature(

DmConversationHasEmptyTimeline,

dmConversationHasEmptyTimeline(dmConversationId, viewerIdOpt))

.withFeature(

DmConversationHasValidLastReadableEventId,

dmConversationHasValidLastReadableEventId(dmConversationId, viewerIdOpt))

.withFeature(

DmConversationInfoExists,

dmConversationInfoExists(dmConversationId, viewerIdOpt))

.withFeature(

DmConversationTimelineExists,

dmConversationTimelineExists(dmConversationId, viewerIdOpt))

.withFeature(

AuthorIsSuspended,

dmConversationHasSuspendedParticipant(dmConversationId, viewerIdOpt))

.withFeature(

AuthorIsDeactivated,

dmConversationHasDeactivatedParticipant(dmConversationId, viewerIdOpt))

.withFeature(

AuthorIsErased,

dmConversationHasErasedParticipant(dmConversationId, viewerIdOpt))

.withFeature(

ViewerIsDmConversationParticipant,

viewerIsDmConversationParticipant(dmConversationId, viewerIdOpt))

def dmConversationIsOneToOneConversation(

dmConversationId: DmConversationId,

viewerIdOpt: Option[UserId]

): Stitch[Boolean] =

viewerIdOpt match {

case Some(viewerId) =>

dmConversationSource.getConversationType(dmConversationId, viewerId).flatMap {

case Some(ConversationType.OneToOneDm | ConversationType.SecretOneToOneDm) =>

Stitch.True

case None =>

Stitch.exception(InvalidDmConversationFeatureException("Conversation type not found"))

case \_ => Stitch.False

}

case \_ => Stitch.exception(InvalidDmConversationFeatureException("Viewer id missing"))

}

private[dms] def dmConversationHasEmptyTimeline(

dmConversationId: DmConversationId,

viewerIdOpt: Option[UserId]

): Stitch[Boolean] =

dmConversationSource

.getConversationTimelineEntries(

dmConversationId,

ConversationQuery(

conversationId = Some(dmConversationId),

options = Some(

ConversationQueryOptions(

perspectivalUserId = viewerIdOpt,

hydrateEvents = Some(false),

supportsReactions = Some(true)

)

),

maxCount = 10

)

).map(\_.forall(entries => entries.isEmpty))

private[dms] def dmConversationHasValidLastReadableEventId(

dmConversationId: DmConversationId,

viewerIdOpt: Option[UserId]

): Stitch[Boolean] =

viewerIdOpt match {

case Some(viewerId) =>

dmConversationSource

.getConversationLastReadableEventId(dmConversationId, viewerId).map(\_.exists(id =>

id > 0L))

case \_ => Stitch.exception(InvalidDmConversationFeatureException("Viewer id missing"))

}

private[dms] def dmConversationInfoExists(

dmConversationId: DmConversationId,

viewerIdOpt: Option[UserId]

): Stitch[Boolean] =

viewerIdOpt match {

case Some(viewerId) =>

dmConversationSource

.getDmConversationInfo(dmConversationId, viewerId).map(\_.isDefined)

case \_ => Stitch.exception(InvalidDmConversationFeatureException("Viewer id missing"))

}

private[dms] def dmConversationTimelineExists(

dmConversationId: DmConversationId,

viewerIdOpt: Option[UserId]

): Stitch[Boolean] =

dmConversationSource

.getConversationTimelineState(

dmConversationId,

ConversationQuery(

conversationId = Some(dmConversationId),

options = Some(

ConversationQueryOptions(

perspectivalUserId = viewerIdOpt,

hydrateEvents = Some(false),

supportsReactions = Some(true)

)

),

maxCount = 1

)

).map {

case Some(TimelineLookupState.NotFound) | None => false

case \_ => true

}

private[dms] def anyConversationParticipantMatchesCondition(

condition: UserId => Stitch[Boolean],

dmConversationId: DmConversationId,

viewerIdOpt: Option[UserId]

): Stitch[Boolean] =

viewerIdOpt match {

case Some(viewerId) =>

dmConversationSource

.getConversationParticipantIds(dmConversationId, viewerId).flatMap {

case Some(participants) =>

Stitch

.collect(participants.map(condition)).map(\_.contains(true)).rescue {

case NotFound =>

Stitch.exception(InvalidDmConversationFeatureException("User not found"))

}

case \_ => Stitch.False

}

case \_ => Stitch.exception(InvalidDmConversationFeatureException("Viewer id missing"))

}

def dmConversationHasSuspendedParticipant(

dmConversationId: DmConversationId,

viewerIdOpt: Option[UserId]

): Stitch[Boolean] =

anyConversationParticipantMatchesCondition(

participant => authorFeatures.authorIsSuspended(participant),

dmConversationId,

viewerIdOpt)

def dmConversationHasDeactivatedParticipant(

dmConversationId: DmConversationId,

viewerIdOpt: Option[UserId]

): Stitch[Boolean] =

anyConversationParticipantMatchesCondition(

participant => authorFeatures.authorIsDeactivated(participant),

dmConversationId,

viewerIdOpt)

def dmConversationHasErasedParticipant(

dmConversationId: DmConversationId,

viewerIdOpt: Option[UserId]

): Stitch[Boolean] =

anyConversationParticipantMatchesCondition(

participant => authorFeatures.authorIsErased(participant),

dmConversationId,

viewerIdOpt)

def viewerIsDmConversationParticipant(

dmConversationId: DmConversationId,

viewerIdOpt: Option[UserId]

): Stitch[Boolean] =

viewerIdOpt match {

case Some(viewerId) =>

dmConversationSource

.getConversationParticipantIds(dmConversationId, viewerId).map {

case Some(participants) => participants.contains(viewerId)

case \_ => false

}

case \_ => Stitch.exception(InvalidDmConversationFeatureException("Viewer id missing"))

}

}