package com.twitter.visibility.builder.dms

import com.twitter.convosvc.thriftscala.Event

import com.twitter.convosvc.thriftscala.StoredDelete

import com.twitter.convosvc.thriftscala.StoredPerspectivalMessageInfo

import com.twitter.convosvc.thriftscala.PerspectivalSpamState

import com.twitter.stitch.Stitch

import com.twitter.visibility.builder.FeatureMapBuilder

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

import com.twitter.visibility.common.DmEventId

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

import com.twitter.visibility.common.UserId

import com.twitter.convosvc.thriftscala.EventType

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.stitch.NotFound

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

import com.twitter.visibility.features.\_

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

class DmEventFeatures(

dmEventSource: DmEventSource,

dmConversationSource: DmConversationSource,

authorFeatures: AuthorFeatures,

dmConversationFeatures: DmConversationFeatures,

statsReceiver: StatsReceiver) {

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

private[this] val requests = scopedStatsReceiver.counter("requests")

def forDmEventId(

dmEventId: DmEventId,

viewerId: UserId

): FeatureMapBuilder => FeatureMapBuilder = {

requests.incr()

val dmEventStitchRef: Stitch[Option[Event]] =

Stitch.ref(dmEventSource.getDmEvent(dmEventId, viewerId))

\_.withFeature(

DmEventIsMessageCreateEvent,

isDmEventType(dmEventStitchRef, EventType.MessageCreate))

.withFeature(

AuthorIsSuspended,

messageCreateEventHasInactiveInitiatingUser(

dmEventStitchRef,

initiatingUser => authorFeatures.authorIsSuspended(initiatingUser))

)

.withFeature(

AuthorIsDeactivated,

messageCreateEventHasInactiveInitiatingUser(

dmEventStitchRef,

initiatingUser => authorFeatures.authorIsDeactivated(initiatingUser))

)

.withFeature(

AuthorIsErased,

messageCreateEventHasInactiveInitiatingUser(

dmEventStitchRef,

initiatingUser => authorFeatures.authorIsErased(initiatingUser))

)

.withFeature(

DmEventOccurredBeforeLastClearedEvent,

dmEventOccurredBeforeLastClearedEvent(dmEventStitchRef, dmEventId, viewerId)

)

.withFeature(

DmEventOccurredBeforeJoinConversationEvent,

dmEventOccurredBeforeJoinConversationEvent(dmEventStitchRef, dmEventId, viewerId)

)

.withFeature(

ViewerIsDmConversationParticipant,

dmEventViewerIsDmConversationParticipant(dmEventStitchRef, viewerId)

)

.withFeature(

DmEventIsDeleted,

dmEventIsDeleted(dmEventStitchRef, dmEventId)

)

.withFeature(

DmEventIsHidden,

dmEventIsHidden(dmEventStitchRef, dmEventId)

)

.withFeature(

ViewerIsDmEventInitiatingUser,

viewerIsDmEventInitiatingUser(dmEventStitchRef, viewerId)

)

.withFeature(

DmEventInOneToOneConversationWithUnavailableUser,

dmEventInOneToOneConversationWithUnavailableUser(dmEventStitchRef, viewerId)

)

.withFeature(

DmEventIsLastMessageReadUpdateEvent,

isDmEventType(dmEventStitchRef, EventType.LastMessageReadUpdate)

)

.withFeature(

DmEventIsJoinConversationEvent,

isDmEventType(dmEventStitchRef, EventType.JoinConversation)

)

.withFeature(

DmEventIsWelcomeMessageCreateEvent,

isDmEventType(dmEventStitchRef, EventType.WelcomeMessageCreate)

)

.withFeature(

DmEventIsTrustConversationEvent,

isDmEventType(dmEventStitchRef, EventType.TrustConversation)

)

.withFeature(

DmEventIsCsFeedbackSubmitted,

isDmEventType(dmEventStitchRef, EventType.CsFeedbackSubmitted)

)

.withFeature(

DmEventIsCsFeedbackDismissed,

isDmEventType(dmEventStitchRef, EventType.CsFeedbackDismissed)

)

.withFeature(

DmEventIsConversationCreateEvent,

isDmEventType(dmEventStitchRef, EventType.ConversationCreate)

)

.withFeature(

DmEventInOneToOneConversation,

dmEventInOneToOneConversation(dmEventStitchRef, viewerId)

)

.withFeature(

DmEventIsPerspectivalJoinConversationEvent,

dmEventIsPerspectivalJoinConversationEvent(dmEventStitchRef, dmEventId, viewerId))

}

private def isDmEventType(

dmEventOptStitch: Stitch[Option[Event]],

eventType: EventType

): Stitch[Boolean] =

dmEventSource.getEventType(dmEventOptStitch).flatMap {

case Some(\_: eventType.type) =>

Stitch.True

case None =>

Stitch.exception(InvalidDmEventFeatureException(s"$eventType event type not found"))

case \_ =>

Stitch.False

}

private def dmEventIsPerspectivalJoinConversationEvent(

dmEventOptStitch: Stitch[Option[Event]],

dmEventId: DmEventId,

viewerId: UserId

): Stitch[Boolean] =

Stitch

.join(

dmEventSource.getEventType(dmEventOptStitch),

dmEventSource.getConversationId(dmEventOptStitch)).flatMap {

case (Some(EventType.JoinConversation), conversationIdOpt) =>

conversationIdOpt match {

case Some(conversationId) =>

dmConversationSource

.getParticipantJoinConversationEventId(conversationId, viewerId, viewerId)

.flatMap {

case Some(joinConversationEventId) =>

Stitch.value(joinConversationEventId == dmEventId)

case \_ => Stitch.False

}

case \_ =>

Stitch.exception(InvalidDmEventFeatureException("Conversation id not found"))

}

case (None, \_) =>

Stitch.exception(InvalidDmEventFeatureException("Event type not found"))

case \_ => Stitch.False

}

private def messageCreateEventHasInactiveInitiatingUser(

dmEventOptStitch: Stitch[Option[Event]],

condition: UserId => Stitch[Boolean],

): Stitch[Boolean] =

Stitch

.join(

dmEventSource.getEventType(dmEventOptStitch),

dmEventSource.getInitiatingUserId(dmEventOptStitch)).flatMap {

case (Some(EventType.MessageCreate), Some(userId)) =>

condition(userId).rescue {

case NotFound =>

Stitch.exception(InvalidDmEventFeatureException("initiating user not found"))

}

case (None, \_) =>

Stitch.exception(InvalidDmEventFeatureException("DmEvent type is missing"))

case (Some(EventType.MessageCreate), \_) =>

Stitch.exception(InvalidDmEventFeatureException("initiating user id is missing"))

case \_ => Stitch.False

}

private def dmEventOccurredBeforeLastClearedEvent(

dmEventOptStitch: Stitch[Option[Event]],

dmEventId: DmEventId,

viewerId: UserId

): Stitch[Boolean] = {

dmEventSource.getConversationId(dmEventOptStitch).flatMap {

case Some(convoId) =>

val lastClearedEventIdStitch =

dmConversationSource.getParticipantLastClearedEventId(convoId, viewerId, viewerId)

lastClearedEventIdStitch.flatMap {

case Some(lastClearedEventId) => Stitch(dmEventId <= lastClearedEventId)

case \_ =>

Stitch.False

}

case \_ => Stitch.False

}

}

private def dmEventOccurredBeforeJoinConversationEvent(

dmEventOptStitch: Stitch[Option[Event]],

dmEventId: DmEventId,

viewerId: UserId

): Stitch[Boolean] = {

dmEventSource.getConversationId(dmEventOptStitch).flatMap {

case Some(convoId) =>

val joinConversationEventIdStitch =

dmConversationSource

.getParticipantJoinConversationEventId(convoId, viewerId, viewerId)

joinConversationEventIdStitch.flatMap {

case Some(joinConversationEventId) => Stitch(dmEventId < joinConversationEventId)

case \_ => Stitch.False

}

case \_ => Stitch.False

}

}

private def dmEventViewerIsDmConversationParticipant(

dmEventOptStitch: Stitch[Option[Event]],

viewerId: UserId

): Stitch[Boolean] = {

dmEventSource.getConversationId(dmEventOptStitch).flatMap {

case Some(convoId) =>

dmConversationFeatures.viewerIsDmConversationParticipant(convoId, Some(viewerId))

case \_ => Stitch.True

}

}

private def dmEventIsDeleted(

dmEventOptStitch: Stitch[Option[Event]],

dmEventId: DmEventId

): Stitch[Boolean] =

dmEventSource.getConversationId(dmEventOptStitch).flatMap {

case Some(convoId) =>

dmConversationSource

.getDeleteInfo(convoId, dmEventId).rescue {

case e: java.lang.IllegalArgumentException =>

Stitch.exception(InvalidDmEventFeatureException("Invalid conversation id"))

}.flatMap {

case Some(StoredDelete(None)) => Stitch.True

case \_ => Stitch.False

}

case \_ => Stitch.False

}

private def dmEventIsHidden(

dmEventOptStitch: Stitch[Option[Event]],

dmEventId: DmEventId

): Stitch[Boolean] =

dmEventSource.getConversationId(dmEventOptStitch).flatMap {

case Some(convoId) =>

dmConversationSource

.getPerspectivalMessageInfo(convoId, dmEventId).rescue {

case e: java.lang.IllegalArgumentException =>

Stitch.exception(InvalidDmEventFeatureException("Invalid conversation id"))

}.flatMap {

case Some(StoredPerspectivalMessageInfo(Some(hidden), \_)) if hidden =>

Stitch.True

case Some(StoredPerspectivalMessageInfo(\_, Some(spamState)))

if spamState == PerspectivalSpamState.Spam =>

Stitch.True

case \_ => Stitch.False

}

case \_ => Stitch.False

}

private def viewerIsDmEventInitiatingUser(

dmEventOptStitch: Stitch[Option[Event]],

viewerId: UserId

): Stitch[Boolean] =

Stitch

.join(

dmEventSource.getEventType(dmEventOptStitch),

dmEventSource.getInitiatingUserId(dmEventOptStitch)).flatMap {

case (

Some(

EventType.TrustConversation | EventType.CsFeedbackSubmitted |

EventType.CsFeedbackDismissed | EventType.WelcomeMessageCreate |

EventType.JoinConversation),

Some(userId)) =>

Stitch(viewerId == userId)

case (

Some(

EventType.TrustConversation | EventType.CsFeedbackSubmitted |

EventType.CsFeedbackDismissed | EventType.WelcomeMessageCreate |

EventType.JoinConversation),

None) =>

Stitch.exception(InvalidDmEventFeatureException("Initiating user id is missing"))

case (None, \_) =>

Stitch.exception(InvalidDmEventFeatureException("DmEvent type is missing"))

case \_ => Stitch.True

}

private def dmEventInOneToOneConversationWithUnavailableUser(

dmEventOptStitch: Stitch[Option[Event]],

viewerId: UserId

): Stitch[Boolean] =

dmEventSource.getConversationId(dmEventOptStitch).flatMap {

case Some(conversationId) =>

dmConversationFeatures

.dmConversationIsOneToOneConversation(conversationId, Some(viewerId)).flatMap {

isOneToOne =>

if (isOneToOne) {

Stitch

.join(

dmConversationFeatures

.dmConversationHasSuspendedParticipant(conversationId, Some(viewerId)),

dmConversationFeatures

.dmConversationHasDeactivatedParticipant(conversationId, Some(viewerId)),

dmConversationFeatures

.dmConversationHasErasedParticipant(conversationId, Some(viewerId))

).flatMap {

case (

convoParticipantIsSuspended,

convoParticipantIsDeactivated,

convoParticipantIsErased) =>

Stitch.value(

convoParticipantIsSuspended || convoParticipantIsDeactivated || convoParticipantIsErased)

}

} else {

Stitch.False

}

}

case \_ => Stitch.False

}

private def dmEventInOneToOneConversation(

dmEventOptStitch: Stitch[Option[Event]],

viewerId: UserId

): Stitch[Boolean] =

dmEventSource.getConversationId(dmEventOptStitch).flatMap {

case Some(conversationId) =>

dmConversationFeatures

.dmConversationIsOneToOneConversation(conversationId, Some(viewerId))

case \_ => Stitch.False

}

}