package com.twitter.frigate.pushservice.send\_handler

import com.twitter.escherbird.metadata.thriftscala.EntityMegadata

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.\_

import com.twitter.frigate.common.store.interests.InterestsLookupRequestWithContext

import com.twitter.frigate.common.util.MrNtabCopyObjects

import com.twitter.frigate.common.util.MrPushCopyObjects

import com.twitter.frigate.magic\_events.thriftscala.FanoutEvent

import com.twitter.frigate.pushservice.model.PushTypes.PushCandidate

import com.twitter.frigate.pushservice.model.PushTypes.RawCandidate

import com.twitter.frigate.pushservice.ml.PushMLModelScorer

import com.twitter.frigate.pushservice.model.candidate.CopyIds

import com.twitter.frigate.pushservice.store.EventRequest

import com.twitter.frigate.pushservice.store.UttEntityHydrationStore

import com.twitter.frigate.pushservice.util.CandidateHydrationUtil.\_

import com.twitter.frigate.thriftscala.CommonRecommendationType

import com.twitter.gizmoduck.thriftscala.User

import com.twitter.hermit.store.semantic\_core.SemanticEntityForQuery

import com.twitter.interests.thriftscala.UserInterests

import com.twitter.livevideo.timeline.domain.v2.{Event => LiveEvent}

import com.twitter.simclusters\_v2.thriftscala.SimClustersInferredEntities

import com.twitter.storehaus.ReadableStore

import com.twitter.strato.client.UserId

import com.twitter.ubs.thriftscala.AudioSpace

import com.twitter.util.Future

case class SendHandlerPushCandidateHydrator(

lexServiceStore: ReadableStore[EventRequest, LiveEvent],

fanoutMetadataStore: ReadableStore[(Long, Long), FanoutEvent],

semanticCoreMegadataStore: ReadableStore[SemanticEntityForQuery, EntityMegadata],

safeUserStore: ReadableStore[Long, User],

simClusterToEntityStore: ReadableStore[Int, SimClustersInferredEntities],

audioSpaceStore: ReadableStore[String, AudioSpace],

interestsLookupStore: ReadableStore[InterestsLookupRequestWithContext, UserInterests],

uttEntityHydrationStore: UttEntityHydrationStore,

superFollowCreatorTweetCountStore: ReadableStore[UserId, Int]

)(

implicit statsReceiver: StatsReceiver,

implicit val weightedOpenOrNtabClickModelScorer: PushMLModelScorer) {

lazy val candidateWithCopyNumStat = statsReceiver.stat("candidate\_with\_copy\_num")

lazy val hydratedCandidateStat = statsReceiver.scope("hydrated\_candidates")

def updateCandidates(

candidateDetails: Seq[CandidateDetails[RawCandidate]],

): Future[Seq[CandidateDetails[PushCandidate]]] = {

Future.collect {

candidateDetails.map { candidateDetail =>

val pushCandidate = candidateDetail.candidate

val copyIds = getCopyIdsByCRT(pushCandidate.commonRecType)

val hydratedCandidateFut = pushCandidate match {

case magicFanoutNewsEventCandidate: MagicFanoutNewsEventCandidate =>

getHydratedCandidateForMagicFanoutNewsEvent(

magicFanoutNewsEventCandidate,

copyIds,

lexServiceStore,

fanoutMetadataStore,

semanticCoreMegadataStore,

simClusterToEntityStore,

interestsLookupStore,

uttEntityHydrationStore

)

case scheduledSpaceSubscriberCandidate: ScheduledSpaceSubscriberCandidate =>

getHydratedCandidateForScheduledSpaceSubscriber(

scheduledSpaceSubscriberCandidate,

safeUserStore,

copyIds,

audioSpaceStore

)

case scheduledSpaceSpeakerCandidate: ScheduledSpaceSpeakerCandidate =>

getHydratedCandidateForScheduledSpaceSpeaker(

scheduledSpaceSpeakerCandidate,

safeUserStore,

copyIds,

audioSpaceStore

)

case magicFanoutSportsEventCandidate: MagicFanoutSportsEventCandidate with MagicFanoutSportsScoreInformation =>

getHydratedCandidateForMagicFanoutSportsEvent(

magicFanoutSportsEventCandidate,

copyIds,

lexServiceStore,

fanoutMetadataStore,

semanticCoreMegadataStore,

interestsLookupStore,

uttEntityHydrationStore

)

case magicFanoutProductLaunchCandidate: MagicFanoutProductLaunchCandidate =>

getHydratedCandidateForMagicFanoutProductLaunch(

magicFanoutProductLaunchCandidate,

copyIds)

case creatorEventCandidate: MagicFanoutCreatorEventCandidate =>

getHydratedCandidateForMagicFanoutCreatorEvent(

creatorEventCandidate,

safeUserStore,

copyIds,

superFollowCreatorTweetCountStore)

case \_ =>

throw new IllegalArgumentException("Incorrect candidate type when update candidates")

}

hydratedCandidateFut.map { hydratedCandidate =>

hydratedCandidateStat.counter(hydratedCandidate.commonRecType.name).incr()

CandidateDetails(

hydratedCandidate,

source = candidateDetail.source

)

}

}

}

}

private def getCopyIdsByCRT(crt: CommonRecommendationType): CopyIds = {

crt match {

case CommonRecommendationType.MagicFanoutNewsEvent =>

CopyIds(

pushCopyId = Some(MrPushCopyObjects.MagicFanoutNewsPushCopy.copyId),

ntabCopyId = Some(MrNtabCopyObjects.MagicFanoutNewsForYouCopy.copyId),

aggregationId = None

)

case CommonRecommendationType.ScheduledSpaceSubscriber =>

CopyIds(

pushCopyId = Some(MrPushCopyObjects.ScheduledSpaceSubscriber.copyId),

ntabCopyId = Some(MrNtabCopyObjects.ScheduledSpaceSubscriber.copyId),

aggregationId = None

)

case CommonRecommendationType.ScheduledSpaceSpeaker =>

CopyIds(

pushCopyId = Some(MrPushCopyObjects.ScheduledSpaceSpeaker.copyId),

ntabCopyId = Some(MrNtabCopyObjects.ScheduledSpaceSpeakerNow.copyId),

aggregationId = None

)

case CommonRecommendationType.SpaceSpeaker =>

CopyIds(

pushCopyId = Some(MrPushCopyObjects.SpaceSpeaker.copyId),

ntabCopyId = Some(MrNtabCopyObjects.SpaceSpeaker.copyId),

aggregationId = None

)

case CommonRecommendationType.SpaceHost =>

CopyIds(

pushCopyId = Some(MrPushCopyObjects.SpaceHost.copyId),

ntabCopyId = Some(MrNtabCopyObjects.SpaceHost.copyId),

aggregationId = None

)

case CommonRecommendationType.MagicFanoutSportsEvent =>

CopyIds(

pushCopyId = Some(MrPushCopyObjects.MagicFanoutSportsPushCopy.copyId),

ntabCopyId = Some(MrNtabCopyObjects.MagicFanoutSportsCopy.copyId),

aggregationId = None

)

case CommonRecommendationType.MagicFanoutProductLaunch =>

CopyIds(

pushCopyId = Some(MrPushCopyObjects.MagicFanoutProductLaunch.copyId),

ntabCopyId = Some(MrNtabCopyObjects.ProductLaunch.copyId),

aggregationId = None

)

case CommonRecommendationType.CreatorSubscriber =>

CopyIds(

pushCopyId = Some(MrPushCopyObjects.MagicFanoutCreatorSubscription.copyId),

ntabCopyId = Some(MrNtabCopyObjects.MagicFanoutCreatorSubscription.copyId),

aggregationId = None

)

case CommonRecommendationType.NewCreator =>

CopyIds(

pushCopyId = Some(MrPushCopyObjects.MagicFanoutNewCreator.copyId),

ntabCopyId = Some(MrNtabCopyObjects.MagicFanoutNewCreator.copyId),

aggregationId = None

)

case \_ =>

throw new IllegalArgumentException("Incorrect candidate type when fetch copy ids")

}

}

def apply(

candidateDetails: Seq[CandidateDetails[RawCandidate]]

): Future[Seq[CandidateDetails[PushCandidate]]] = {

updateCandidates(candidateDetails)

}

}