package com.twitter.timelineranker.entity\_tweets

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.servo.util.FutureArrow

import com.twitter.storehaus.Store

import com.twitter.timelineranker.common.\_

import com.twitter.timelineranker.core.HydratedCandidatesAndFeaturesEnvelope

import com.twitter.timelineranker.model.RecapQuery.DependencyProvider

import com.twitter.timelineranker.model.\_

import com.twitter.timelineranker.parameters.entity\_tweets.EntityTweetsParams.\_

import com.twitter.timelineranker.recap.model.ContentFeatures

import com.twitter.timelineranker.util.CopyContentFeaturesIntoHydratedTweetsTransform

import com.twitter.timelineranker.util.CopyContentFeaturesIntoThriftTweetFeaturesTransform

import com.twitter.timelineranker.util.TweetFilters

import com.twitter.timelineranker.visibility.FollowGraphDataProvider

import com.twitter.timelines.clients.gizmoduck.GizmoduckClient

import com.twitter.timelines.clients.manhattan.UserMetadataClient

import com.twitter.timelines.clients.relevance\_search.SearchClient

import com.twitter.timelines.clients.tweetypie.TweetyPieClient

import com.twitter.timelines.model.TweetId

import com.twitter.timelines.util.FailOpenHandler

import com.twitter.timelines.util.stats.RequestStatsReceiver

import com.twitter.timelines.visibility.VisibilityEnforcer

import com.twitter.util.Future

class EntityTweetsSource(

gizmoduckClient: GizmoduckClient,

searchClient: SearchClient,

tweetyPieClient: TweetyPieClient,

userMetadataClient: UserMetadataClient,

followGraphDataProvider: FollowGraphDataProvider,

visibilityEnforcer: VisibilityEnforcer,

contentFeaturesCache: Store[TweetId, ContentFeatures],

statsReceiver: StatsReceiver) {

private[this] val baseScope = statsReceiver.scope("entityTweetsSource")

private[this] val requestStats = RequestStatsReceiver(baseScope)

private[this] val failOpenScope = baseScope.scope("failOpen")

private[this] val userProfileHandler = new FailOpenHandler(failOpenScope, "userProfileInfo")

private[this] val userLanguagesHandler = new FailOpenHandler(failOpenScope, "userLanguages")

private[this] val followGraphDataTransform = new FollowGraphDataTransform(

followGraphDataProvider = followGraphDataProvider,

maxFollowedUsersProvider = DependencyProvider.from(MaxFollowedUsersParam)

)

private[this] val fetchSearchResultsTransform = new EntityTweetsSearchResultsTransform(

searchClient = searchClient,

statsReceiver = baseScope

)

private[this] val userProfileInfoTransform =

new UserProfileInfoTransform(userProfileHandler, gizmoduckClient)

private[this] val languagesTransform =

new UserLanguagesTransform(userLanguagesHandler, userMetadataClient)

private[this] val visibilityEnforcingTransform = new VisibilityEnforcingTransform(

visibilityEnforcer

)

private[this] val filters = TweetFilters.ValueSet(

TweetFilters.DuplicateTweets,

TweetFilters.DuplicateRetweets

)

private[this] val hydratedTweetsFilter = new HydratedTweetsFilterTransform(

outerFilters = filters,

innerFilters = TweetFilters.None,

useFollowGraphData = false,

useSourceTweets = false,

statsReceiver = baseScope,

numRetweetsAllowed = HydratedTweetsFilterTransform.NumDuplicateRetweetsAllowed

)

private[this] val contentFeaturesHydrationTransform =

new ContentFeaturesHydrationTransformBuilder(

tweetyPieClient = tweetyPieClient,

contentFeaturesCache = contentFeaturesCache,

enableContentFeaturesGate = RecapQuery.paramGate(EnableContentFeaturesHydrationParam),

enableTokensInContentFeaturesGate =

RecapQuery.paramGate(EnableTokensInContentFeaturesHydrationParam),

enableTweetTextInContentFeaturesGate =

RecapQuery.paramGate(EnableTweetTextInContentFeaturesHydrationParam),

enableConversationControlContentFeaturesGate =

RecapQuery.paramGate(EnableConversationControlInContentFeaturesHydrationParam),

enableTweetMediaHydrationGate = RecapQuery.paramGate(EnableTweetMediaHydrationParam),

hydrateInReplyToTweets = false,

statsReceiver = baseScope

).build()

private[this] def hydratesContentFeatures(

hydratedEnvelope: HydratedCandidatesAndFeaturesEnvelope

): Boolean =

hydratedEnvelope.candidateEnvelope.query.hydratesContentFeatures.getOrElse(true)

private[this] val contentFeaturesTransformer = FutureArrow.choose(

predicate = hydratesContentFeatures,

ifTrue = contentFeaturesHydrationTransform

.andThen(CopyContentFeaturesIntoThriftTweetFeaturesTransform)

.andThen(CopyContentFeaturesIntoHydratedTweetsTransform),

ifFalse = FutureArrow[

HydratedCandidatesAndFeaturesEnvelope,

HydratedCandidatesAndFeaturesEnvelope

](Future.value) // empty transformer

)

private[this] val candidateGenerationTransform = new CandidateGenerationTransform(baseScope)

private[this] val hydrationAndFilteringPipeline =

CreateCandidateEnvelopeTransform

.andThen(followGraphDataTransform) // Fetch follow graph data

.andThen(fetchSearchResultsTransform) // fetch search results

.andThen(SearchResultDedupAndSortingTransform) // dedup and order search results

.andThen(CandidateTweetHydrationTransform) // hydrate search results

.andThen(visibilityEnforcingTransform) // filter hydrated tweets to visible ones

.andThen(hydratedTweetsFilter) // filter hydrated tweets based on predefined filter

.andThen(

TrimToMatchHydratedTweetsTransform

) // trim search result set to match filtered hydrated tweets (this needs to be accurate for feature hydration)

// runs the main pipeline in parallel with fetching user profile info and user languages

private[this] val featureHydrationDataTransform =

new FeatureHydrationDataTransform(

hydrationAndFilteringPipeline,

languagesTransform,

userProfileInfoTransform

)

private[this] val tweetFeaturesHydrationTransform =

OutOfNetworkTweetsSearchFeaturesHydrationTransform

.andThen(contentFeaturesTransformer)

private[this] val featureHydrationPipeline =

featureHydrationDataTransform

.andThen(tweetFeaturesHydrationTransform)

.andThen(candidateGenerationTransform)

def get(query: RecapQuery): Future[CandidateTweetsResult] = {

requestStats.addEventStats {

featureHydrationPipeline(query)

}

}

def get(queries: Seq[RecapQuery]): Future[Seq[CandidateTweetsResult]] = {

Future.collect(queries.map(get))

}

}