package com.twitter.follow\_recommendations.flows.content\_recommender\_flow

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.base.EnrichedCandidateSource

import com.twitter.follow\_recommendations.common.base.GatedPredicateBase

import com.twitter.follow\_recommendations.common.base.ParamPredicate

import com.twitter.follow\_recommendations.common.base.Predicate

import com.twitter.follow\_recommendations.common.base.Ranker

import com.twitter.follow\_recommendations.common.base.RecommendationFlow

import com.twitter.follow\_recommendations.common.base.RecommendationResultsConfig

import com.twitter.follow\_recommendations.common.base.Transform

import com.twitter.follow\_recommendations.common.models.CandidateUser

import com.twitter.follow\_recommendations.common.predicates.ExcludedUserIdPredicate

import com.twitter.follow\_recommendations.common.predicates.InactivePredicate

import com.twitter.follow\_recommendations.common.predicates.gizmoduck.GizmoduckPredicate

import com.twitter.follow\_recommendations.common.predicates.sgs.InvalidRelationshipPredicate

import com.twitter.follow\_recommendations.common.predicates.sgs.InvalidTargetCandidateRelationshipTypesPredicate

import com.twitter.follow\_recommendations.common.predicates.sgs.RecentFollowingPredicate

import com.twitter.follow\_recommendations.common.rankers.weighted\_candidate\_source\_ranker.WeightedCandidateSourceRanker

import com.twitter.follow\_recommendations.common.transforms.dedup.DedupTransform

import com.twitter.follow\_recommendations.common.transforms.tracking\_token.TrackingTokenTransform

import com.twitter.follow\_recommendations.utils.CandidateSourceHoldbackUtil

import com.twitter.follow\_recommendations.utils.RecommendationFlowBaseSideEffectsUtil

import com.twitter.product\_mixer.core.functional\_component.candidate\_source.CandidateSource

import com.twitter.product\_mixer.core.quality\_factor.BoundsWithDefault

import com.twitter.product\_mixer.core.quality\_factor.LinearLatencyQualityFactor

import com.twitter.product\_mixer.core.quality\_factor.LinearLatencyQualityFactorConfig

import com.twitter.product\_mixer.core.quality\_factor.LinearLatencyQualityFactorObserver

import com.twitter.product\_mixer.core.quality\_factor.QualityFactorObserver

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class ContentRecommenderFlow @Inject() (

contentRecommenderFlowCandidateSourceRegistry: ContentRecommenderFlowCandidateSourceRegistry,

recentFollowingPredicate: RecentFollowingPredicate,

gizmoduckPredicate: GizmoduckPredicate,

inactivePredicate: InactivePredicate,

sgsPredicate: InvalidTargetCandidateRelationshipTypesPredicate,

invalidRelationshipPredicate: InvalidRelationshipPredicate,

trackingTokenTransform: TrackingTokenTransform,

baseStatsReceiver: StatsReceiver)

extends RecommendationFlow[ContentRecommenderRequest, CandidateUser]

with RecommendationFlowBaseSideEffectsUtil[ContentRecommenderRequest, CandidateUser]

with CandidateSourceHoldbackUtil {

override val statsReceiver: StatsReceiver = baseStatsReceiver.scope("content\_recommender\_flow")

override val qualityFactorObserver: Option[QualityFactorObserver] = {

val config = LinearLatencyQualityFactorConfig(

qualityFactorBounds =

BoundsWithDefault(minInclusive = 0.1, maxInclusive = 1.0, default = 1.0),

initialDelay = 60.seconds,

targetLatency = 100.milliseconds,

targetLatencyPercentile = 95.0,

delta = 0.001

)

val qualityFactor = LinearLatencyQualityFactor(config)

val observer = LinearLatencyQualityFactorObserver(qualityFactor)

statsReceiver.provideGauge("quality\_factor")(qualityFactor.currentValue.toFloat)

Some(observer)

}

protected override def targetEligibility: Predicate[ContentRecommenderRequest] =

new ParamPredicate[ContentRecommenderRequest](

ContentRecommenderParams.TargetEligibility

)

protected override def candidateSources(

target: ContentRecommenderRequest

): Seq[CandidateSource[ContentRecommenderRequest, CandidateUser]] = {

import EnrichedCandidateSource.\_

val identifiers = ContentRecommenderFlowCandidateSourceWeights.getWeights(target.params).keySet

val selected = contentRecommenderFlowCandidateSourceRegistry.select(identifiers)

val budget =

target.params(ContentRecommenderParams.FetchCandidateSourceBudgetInMillisecond).millisecond

filterCandidateSources(target, selected.map(c => c.failOpenWithin(budget, statsReceiver)).toSeq)

}

protected override val preRankerCandidateFilter: Predicate[

(ContentRecommenderRequest, CandidateUser)

] = {

val preRankerFilterStats = statsReceiver.scope("pre\_ranker")

val recentFollowingPredicateStats = preRankerFilterStats.scope("recent\_following\_predicate")

val invalidRelationshipPredicateStats =

preRankerFilterStats.scope("invalid\_relationship\_predicate")

object recentFollowingGatedPredicate

extends GatedPredicateBase[(ContentRecommenderRequest, CandidateUser)](

recentFollowingPredicate,

recentFollowingPredicateStats

) {

override def gate(item: (ContentRecommenderRequest, CandidateUser)): Boolean =

item.\_1.params(ContentRecommenderParams.EnableRecentFollowingPredicate)

}

object invalidRelationshipGatedPredicate

extends GatedPredicateBase[(ContentRecommenderRequest, CandidateUser)](

invalidRelationshipPredicate,

invalidRelationshipPredicateStats

) {

override def gate(item: (ContentRecommenderRequest, CandidateUser)): Boolean =

item.\_1.params(ContentRecommenderParams.EnableInvalidRelationshipPredicate)

}

ExcludedUserIdPredicate

.observe(preRankerFilterStats.scope("exclude\_user\_id\_predicate"))

.andThen(recentFollowingGatedPredicate.observe(recentFollowingPredicateStats))

.andThen(invalidRelationshipGatedPredicate.observe(invalidRelationshipPredicateStats))

}

/\*\*

\* rank the candidates

\*/

protected override def selectRanker(

target: ContentRecommenderRequest

): Ranker[ContentRecommenderRequest, CandidateUser] = {

val rankersStatsReceiver = statsReceiver.scope("rankers")

WeightedCandidateSourceRanker

.build[ContentRecommenderRequest](

ContentRecommenderFlowCandidateSourceWeights.getWeights(target.params),

randomSeed = target.getRandomizationSeed

).observe(rankersStatsReceiver.scope("weighted\_candidate\_source\_ranker"))

}

/\*\*

\* transform the candidates after ranking

\*/

protected override def postRankerTransform: Transform[

ContentRecommenderRequest,

CandidateUser

] = {

new DedupTransform[ContentRecommenderRequest, CandidateUser]

.observe(statsReceiver.scope("dedupping"))

}

protected override def validateCandidates: Predicate[

(ContentRecommenderRequest, CandidateUser)

] = {

val stats = statsReceiver.scope("validate\_candidates")

val gizmoduckPredicateStats = stats.scope("gizmoduck\_predicate")

val inactivePredicateStats = stats.scope("inactive\_predicate")

val sgsPredicateStats = stats.scope("sgs\_predicate")

val includeGizmoduckPredicate =

new ParamPredicate[ContentRecommenderRequest](

ContentRecommenderParams.EnableGizmoduckPredicate)

.map[(ContentRecommenderRequest, CandidateUser)] {

case (request: ContentRecommenderRequest, \_) =>

request

}

val includeInactivePredicate =

new ParamPredicate[ContentRecommenderRequest](

ContentRecommenderParams.EnableInactivePredicate)

.map[(ContentRecommenderRequest, CandidateUser)] {

case (request: ContentRecommenderRequest, \_) =>

request

}

val includeInvalidTargetCandidateRelationshipTypesPredicate =

new ParamPredicate[ContentRecommenderRequest](

ContentRecommenderParams.EnableInvalidTargetCandidateRelationshipPredicate)

.map[(ContentRecommenderRequest, CandidateUser)] {

case (request: ContentRecommenderRequest, \_) =>

request

}

Predicate

.andConcurrently[(ContentRecommenderRequest, CandidateUser)](

Seq(

gizmoduckPredicate.observe(gizmoduckPredicateStats).gate(includeGizmoduckPredicate),

inactivePredicate.observe(inactivePredicateStats).gate(includeInactivePredicate),

sgsPredicate

.observe(sgsPredicateStats).gate(

includeInvalidTargetCandidateRelationshipTypesPredicate),

)

)

}

/\*\*

\* transform the candidates into results and return

\*/

protected override def transformResults: Transform[ContentRecommenderRequest, CandidateUser] = {

trackingTokenTransform

}

/\*\*

\* configuration for recommendation results

\*/

protected override def resultsConfig(

target: ContentRecommenderRequest

): RecommendationResultsConfig = {

RecommendationResultsConfig(

target.maxResults.getOrElse(target.params(ContentRecommenderParams.ResultSizeParam)),

target.params(ContentRecommenderParams.BatchSizeParam)

)

}

}