package com.twitter.follow\_recommendations.flows.ads

import com.twitter.finagle.stats.StatsReceiver

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

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

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

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.base.TruePredicate

import com.twitter.follow\_recommendations.common.candidate\_sources.promoted\_accounts.PromotedAccountsCandidateSource

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

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

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

import com.twitter.inject.annotations.Flag

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

import com.twitter.util.Duration

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class PromotedAccountsFlow @Inject() (

promotedAccountsCandidateSource: PromotedAccountsCandidateSource,

trackingTokenTransform: TrackingTokenTransform,

baseStatsReceiver: StatsReceiver,

@Flag("fetch\_prod\_promoted\_accounts") fetchProductionPromotedAccounts: Boolean)

extends RecommendationFlow[PromotedAccountsFlowRequest, CandidateUser] {

protected override def targetEligibility: Predicate[PromotedAccountsFlowRequest] =

new ParamPredicate[PromotedAccountsFlowRequest](

PromotedAccountsFlowParams.TargetEligibility

)

protected override def candidateSources(

target: PromotedAccountsFlowRequest

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

import EnrichedCandidateSource.\_

val candidateSourceStats = statsReceiver.scope("candidate\_sources")

val budget: Duration = target.params(PromotedAccountsFlowParams.FetchCandidateSourceBudget)

val candidateSources = Seq(

promotedAccountsCandidateSource

.mapKeys[PromotedAccountsFlowRequest](r =>

Seq(r.toAdsRequest(fetchProductionPromotedAccounts)))

.mapValue(PromotedAccountsUtil.toCandidateUser)

).map { candidateSource =>

candidateSource

.failOpenWithin(budget, candidateSourceStats).observe(candidateSourceStats)

}

candidateSources

}

protected override def preRankerCandidateFilter: Predicate[

(PromotedAccountsFlowRequest, CandidateUser)

] = {

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

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

}

/\*\*

\* rank the candidates

\*/

protected override def selectRanker(

target: PromotedAccountsFlowRequest

): Ranker[PromotedAccountsFlowRequest, CandidateUser] = {

new IdentityRanker[PromotedAccountsFlowRequest, CandidateUser]

}

/\*\*

\* transform the candidates after ranking (e.g. dedupping, grouping and etc)

\*/

protected override def postRankerTransform: Transform[

PromotedAccountsFlowRequest,

CandidateUser

] = {

new IdentityTransform[PromotedAccountsFlowRequest, CandidateUser]

}

/\*\*

\* filter invalid candidates before returning the results.

\*

\* Some heavy filters e.g. SGS filter could be applied in this step

\*/

protected override def validateCandidates: Predicate[

(PromotedAccountsFlowRequest, CandidateUser)

] = {

new TruePredicate[(PromotedAccountsFlowRequest, CandidateUser)]

}

/\*\*

\* transform the candidates into results and return

\*/

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

trackingTokenTransform

}

/\*\*

\* configuration for recommendation results

\*/

protected override def resultsConfig(

target: PromotedAccountsFlowRequest

): RecommendationResultsConfig = {

RecommendationResultsConfig(

target.params(PromotedAccountsFlowParams.ResultSizeParam),

target.params(PromotedAccountsFlowParams.BatchSizeParam)

)

}

override val statsReceiver: StatsReceiver = baseStatsReceiver.scope("promoted\_accounts\_flow")

}