package com.twitter.follow\_recommendations.models

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

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

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

import com.twitter.follow\_recommendations.common.utils.UserSignupUtil

import com.twitter.follow\_recommendations.logging.{thriftscala => offline}

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

import com.twitter.product\_mixer.core.model.common.identifier.CandidateSourceIdentifier

import com.twitter.product\_mixer.core.model.common.identifier.RecommendationPipelineIdentifier

import com.twitter.product\_mixer.core.model.marshalling.HasMarshalling

import com.twitter.product\_mixer.core.model.marshalling.request.HasClientContext

import com.twitter.util.Time

case class RecommendationFlowData[Target <: HasClientContext](

request: Target,

recommendationFlowIdentifier: RecommendationPipelineIdentifier,

candidateSources: Seq[CandidateSource[Target, CandidateUser]],

candidatesFromCandidateSources: Seq[CandidateUser],

mergedCandidates: Seq[CandidateUser],

filteredCandidates: Seq[CandidateUser],

rankedCandidates: Seq[CandidateUser],

transformedCandidates: Seq[CandidateUser],

truncatedCandidates: Seq[CandidateUser],

results: Seq[CandidateUser])

extends HasMarshalling {

import RecommendationFlowData.\_

lazy val toRecommendationFlowLogOfflineThrift: offline.RecommendationFlowLog = {

val userMetadata = userToOfflineRecommendationFlowUserMetadata(request)

val signals = userToOfflineRecommendationFlowSignals(request)

val filteredCandidateSourceCandidates =

candidatesToOfflineRecommendationFlowCandidateSourceCandidates(

candidateSources,

filteredCandidates

)

val rankedCandidateSourceCandidates =

candidatesToOfflineRecommendationFlowCandidateSourceCandidates(

candidateSources,

rankedCandidates

)

val truncatedCandidateSourceCandidates =

candidatesToOfflineRecommendationFlowCandidateSourceCandidates(

candidateSources,

truncatedCandidates

)

offline.RecommendationFlowLog(

ClientContextConverter.toFRSOfflineClientContextThrift(request.clientContext),

userMetadata,

signals,

Time.now.inMillis,

recommendationFlowIdentifier.name,

Some(filteredCandidateSourceCandidates),

Some(rankedCandidateSourceCandidates),

Some(truncatedCandidateSourceCandidates)

)

}

}

object RecommendationFlowData {

def userToOfflineRecommendationFlowUserMetadata[Target <: HasClientContext](

request: Target

): Option[offline.OfflineRecommendationFlowUserMetadata] = {

val userSignupAge = UserSignupUtil.userSignupAge(request).map(\_.inDays)

val userState = request match {

case req: HasUserState => req.userState.map(\_.name)

case \_ => None

}

Some(offline.OfflineRecommendationFlowUserMetadata(userSignupAge, userState))

}

def userToOfflineRecommendationFlowSignals[Target <: HasClientContext](

request: Target

): Option[offline.OfflineRecommendationFlowSignals] = {

val countryCode = request.getCountryCode

Some(offline.OfflineRecommendationFlowSignals(countryCode))

}

def candidatesToOfflineRecommendationFlowCandidateSourceCandidates[Target <: HasClientContext](

candidateSources: Seq[CandidateSource[Target, CandidateUser]],

candidates: Seq[CandidateUser],

): Seq[offline.OfflineRecommendationFlowCandidateSourceCandidates] = {

val candidatesGroupedByCandidateSources =

candidates.groupBy(

\_.getPrimaryCandidateSource.getOrElse(CandidateSourceIdentifier("NoCandidateSource")))

candidateSources.map(candidateSource => {

val candidates =

candidatesGroupedByCandidateSources.get(candidateSource.identifier).toSeq.flatten

val candidateUserIds = candidates.map(\_.id)

val candidateUserScores = candidates.map(\_.score).exists(\_.nonEmpty) match {

case true => Some(candidates.map(\_.score.getOrElse(-1.0)))

case false => None

}

offline.OfflineRecommendationFlowCandidateSourceCandidates(

candidateSource.identifier.name,

candidateUserIds,

candidateUserScores

)

})

}

}