package com.twitter.follow\_recommendations.models

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

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

import com.twitter.follow\_recommendations.logging.thriftscala.OfflineDisplayContext

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

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

import scala.reflect.ClassTag

import scala.reflect.classTag

trait DisplayContext {

def toOfflineThrift: offline.OfflineDisplayContext

}

object DisplayContext {

case class Profile(profileId: Long) extends DisplayContext {

override val toOfflineThrift: OfflineDisplayContext =

offline.OfflineDisplayContext.Profile(offline.OfflineProfile(profileId))

}

case class Search(searchQuery: String) extends DisplayContext {

override val toOfflineThrift: OfflineDisplayContext =

offline.OfflineDisplayContext.Search(offline.OfflineSearch(searchQuery))

}

case class Rux(focalAuthorId: Long) extends DisplayContext {

override val toOfflineThrift: OfflineDisplayContext =

offline.OfflineDisplayContext.Rux(offline.OfflineRux(focalAuthorId))

}

case class Topic(topicId: Long) extends DisplayContext {

override val toOfflineThrift: OfflineDisplayContext =

offline.OfflineDisplayContext.Topic(offline.OfflineTopic(topicId))

}

case class ReactiveFollow(followedUserIds: Seq[Long]) extends DisplayContext {

override val toOfflineThrift: OfflineDisplayContext =

offline.OfflineDisplayContext.ReactiveFollow(offline.OfflineReactiveFollow(followedUserIds))

}

case class NuxInterests(flowContext: Option[FlowContext], uttInterestIds: Option[Seq[Long]])

extends DisplayContext {

override val toOfflineThrift: OfflineDisplayContext =

offline.OfflineDisplayContext.NuxInterests(

offline.OfflineNuxInterests(flowContext.map(\_.toOfflineThrift)))

}

case class PostNuxFollowTask(flowContext: Option[FlowContext]) extends DisplayContext {

override val toOfflineThrift: OfflineDisplayContext =

offline.OfflineDisplayContext.PostNuxFollowTask(

offline.OfflinePostNuxFollowTask(flowContext.map(\_.toOfflineThrift)))

}

case class AdCampaignTarget(similarToUserIds: Seq[Long]) extends DisplayContext {

override val toOfflineThrift: OfflineDisplayContext =

offline.OfflineDisplayContext.AdCampaignTarget(

offline.OfflineAdCampaignTarget(similarToUserIds))

}

case class ConnectTab(

byfSeedUserIds: Seq[Long],

similarToUserIds: Seq[Long],

engagedUserIds: Seq[RecentlyEngagedUserId])

extends DisplayContext {

override val toOfflineThrift: OfflineDisplayContext =

offline.OfflineDisplayContext.ConnectTab(

offline.OfflineConnectTab(

byfSeedUserIds,

similarToUserIds,

engagedUserIds.map(user => user.toOfflineThrift)))

}

case class SimilarToUser(similarToUserId: Long) extends DisplayContext {

override val toOfflineThrift: OfflineDisplayContext =

offline.OfflineDisplayContext.SimilarToUser(offline.OfflineSimilarToUser(similarToUserId))

}

def fromThrift(tDisplayContext: t.DisplayContext): DisplayContext = tDisplayContext match {

case t.DisplayContext.Profile(p) => Profile(p.profileId)

case t.DisplayContext.Search(s) => Search(s.searchQuery)

case t.DisplayContext.Rux(r) => Rux(r.focalAuthorId)

case t.DisplayContext.Topic(t) => Topic(t.topicId)

case t.DisplayContext.ReactiveFollow(f) => ReactiveFollow(f.followedUserIds)

case t.DisplayContext.NuxInterests(n) =>

NuxInterests(n.flowContext.map(FlowContext.fromThrift), n.uttInterestIds)

case t.DisplayContext.AdCampaignTarget(a) =>

AdCampaignTarget(a.similarToUserIds)

case t.DisplayContext.ConnectTab(connect) =>

ConnectTab(

connect.byfSeedUserIds,

connect.similarToUserIds,

connect.recentlyEngagedUserIds.map(RecentlyEngagedUserId.fromThrift))

case t.DisplayContext.SimilarToUser(r) =>

SimilarToUser(r.similarToUserId)

case t.DisplayContext.PostNuxFollowTask(p) =>

PostNuxFollowTask(p.flowContext.map(FlowContext.fromThrift))

case t.DisplayContext.UnknownUnionField(t) =>

throw new UnknownDisplayContextException(t.field.name)

}

def getDisplayContextAs[T <: DisplayContext: ClassTag](displayContext: DisplayContext): T =

displayContext match {

case context: T => context

case \_ =>

throw new UnexpectedDisplayContextTypeException(

displayContext,

classTag[T].getClass.getSimpleName)

}

}

class UnknownDisplayContextException(name: String)

extends Exception(s"Unknown DisplayContext in Thrift: ${name}")

class UnexpectedDisplayContextTypeException(displayContext: DisplayContext, expectedType: String)

extends Exception(s"DisplayContext ${displayContext} not of expected type ${expectedType}")