package com.twitter.frigate.pushservice.refresh\_handler.cross

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.\_

import com.twitter.frigate.common.util.MRPushCopy

import com.twitter.frigate.pushservice.model.PushTypes.RawCandidate

import com.twitter.hermit.predicate.Predicate

import com.twitter.util.Future

private[cross] class CopyFilters(statsReceiver: StatsReceiver) {

private val copyPredicates = new CopyPredicates(statsReceiver.scope("copy\_predicate"))

def execute(rawCandidate: RawCandidate, pushCopies: Seq[MRPushCopy]): Future[Seq[MRPushCopy]] = {

val candidateCopyPairs: Seq[CandidateCopyPair] =

pushCopies.map(CandidateCopyPair(rawCandidate, \_))

val compositePredicate: Predicate[CandidateCopyPair] = rawCandidate match {

case \_: F1FirstDegree | \_: OutOfNetworkTweetCandidate | \_: EventCandidate |

\_: TopicProofTweetCandidate | \_: ListPushCandidate | \_: HermitInterestBasedUserFollow |

\_: UserFollowWithoutSocialContextCandidate | \_: DiscoverTwitterCandidate |

\_: TopTweetImpressionsCandidate | \_: TrendTweetCandidate |

\_: SubscribedSearchTweetCandidate | \_: DigestCandidate =>

copyPredicates.alwaysTruePredicate

case \_: SocialContextActions => copyPredicates.displaySocialContextPredicate

case \_ => copyPredicates.unrecognizedCandidatePredicate // block unrecognised candidates

}

// apply predicate to all [[MRPushCopy]] objects

val filterResults: Future[Seq[Boolean]] = compositePredicate(candidateCopyPairs)

filterResults.map { results: Seq[Boolean] =>

val seqBuilder = Seq.newBuilder[MRPushCopy]

results.zip(pushCopies).foreach {

case (result, pushCopy) => if (result) seqBuilder += pushCopy

}

seqBuilder.result()

}

}

}