package com.twitter.product\_mixer.component\_library.selector

import com.twitter.product\_mixer.component\_library.model.candidate.BaseTweetCandidate

import com.twitter.product\_mixer.component\_library.model.candidate.IsPinnedFeature

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMapBuilder

import com.twitter.product\_mixer.core.model.common.presentation.ItemCandidateWithDetails

import com.twitter.product\_mixer.core.model.common.presentation.CandidatePipelines

import com.twitter.product\_mixer.core.model.common.presentation.CandidateSources

import com.twitter.product\_mixer.core.model.common.presentation.CandidateSourcePosition

/\*\*

\* Once a pair of duplicate candidates has been found we need to someone 'resolve' the duplication.

\* This may be as simple as picking whichever candidate came first (see [[PickFirstCandidateMerger]]

\* but this strategy could mean losing important candidate information. Candidates might, for

\* example, have different features. [[CandidateMergeStrategy]] lets you define a custom behavior

\* for resolving duplication to help support these more nuanced situations.

\*/

trait CandidateMergeStrategy {

def apply(

existingCandidate: ItemCandidateWithDetails,

newCandidate: ItemCandidateWithDetails

): ItemCandidateWithDetails

}

/\*\*

\* Keep whichever candidate was encountered first.

\*/

object PickFirstCandidateMerger extends CandidateMergeStrategy {

override def apply(

existingCandidate: ItemCandidateWithDetails,

newCandidate: ItemCandidateWithDetails

): ItemCandidateWithDetails = existingCandidate

}

/\*\*

\* Keep the candidate encountered first but combine all candidate feature maps.

\*/

object CombineFeatureMapsCandidateMerger extends CandidateMergeStrategy {

override def apply(

existingCandidate: ItemCandidateWithDetails,

newCandidate: ItemCandidateWithDetails

): ItemCandidateWithDetails = {

// Prepend new because list set keeps insertion order, and last operations in ListSet are O(1)

val mergedCandidateSourceIdentifiers =

newCandidate.features.get(CandidateSources) ++ existingCandidate.features

.get(CandidateSources)

val mergedCandidatePipelineIdentifiers =

newCandidate.features.get(CandidatePipelines) ++ existingCandidate.features

.get(CandidatePipelines)

// the unitary features are pulled from the existing candidate as explained above, while

// Set Features are merged/accumulated.

val mergedCommonFeatureMap = FeatureMapBuilder()

.add(CandidatePipelines, mergedCandidatePipelineIdentifiers)

.add(CandidateSources, mergedCandidateSourceIdentifiers)

.add(CandidateSourcePosition, existingCandidate.sourcePosition)

.build()

existingCandidate.copy(features =

existingCandidate.features ++ newCandidate.features ++ mergedCommonFeatureMap)

}

}

/\*\*

\* Keep the pinnable candidate. For cases where we are dealing with duplicate entries across

\* different candidate types, such as different sub-classes of

\* [[com.twitter.product\_mixer.component\_library.model.candidate.BaseTweetCandidate]], we will

\* prioritize the candidate with [[IsPinnedFeature]] because it contains additional information

\* needed for the positioning of a pinned entry on a timeline.

\*/

object PickPinnedCandidateMerger extends CandidateMergeStrategy {

override def apply(

existingCandidate: ItemCandidateWithDetails,

newCandidate: ItemCandidateWithDetails

): ItemCandidateWithDetails =

Seq(existingCandidate, newCandidate)

.collectFirst {

case candidate @ ItemCandidateWithDetails(\_: BaseTweetCandidate, \_, features)

if features.getTry(IsPinnedFeature).toOption.contains(true) =>

candidate

}.getOrElse(existingCandidate)

}