package com.twitter.product\_mixer.core.model.common.presentation

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

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

import com.twitter.product\_mixer.core.model.common.CandidateWithFeatures

import com.twitter.product\_mixer.core.model.common.UniversalNoun

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

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.PipelineFailure

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.UnexpectedCandidateResult

import scala.collection.immutable.ListSet

import scala.reflect.ClassTag

sealed trait CandidateWithDetails { self =>

def presentation: Option[UniversalPresentation]

def features: FeatureMap

// last of the set because in ListSet, the last element is the first inserted one with O(1)

// access

lazy val source: CandidatePipelineIdentifier = features.get(CandidatePipelines).last

lazy val sourcePosition: Int = features.get(CandidateSourcePosition)

/\*\*

\* @see [[getCandidateId]]

\*/

def candidateIdLong: Long = getCandidateId[Long]

/\*\*

\* @see [[getCandidateId]]

\*/

def candidateIdString: String = getCandidateId[String]

/\*\*

\* Convenience method for retrieving a candidate ID off of the base [[CandidateWithDetails]] trait

\* without manually pattern matching.

\*

\* @throws PipelineFailure if CandidateIdType does not match the expected Item Candidate Id type,

\* or if invoked on a Module Candidate

\*/

def getCandidateId[CandidateIdType](

)(

implicit tag: ClassTag[CandidateIdType]

): CandidateIdType =

self match {

case item: ItemCandidateWithDetails =>

item.candidate.id match {

case id: CandidateIdType => id

case \_ =>

throw PipelineFailure(

UnexpectedCandidateResult,

s"Invalid Item Candidate ID type expected $tag for Item Candidate type ${item.candidate.getClass}")

}

case \_: ModuleCandidateWithDetails =>

throw PipelineFailure(

UnexpectedCandidateResult,

"Cannot retrieve Item Candidate ID for a Module")

}

/\*\*

\* Convenience method for retrieving a candidate off of the base [[CandidateWithDetails]] trait

\* without manually pattern matching.

\*

\* @throws PipelineFailure if CandidateType does not match the expected Item Candidate type, or

\* if invoked on a Module Candidate

\*/

def getCandidate[CandidateType <: UniversalNoun[\_]](

)(

implicit tag: ClassTag[CandidateType]

): CandidateType =

self match {

case ItemCandidateWithDetails(candidate: CandidateType, \_, \_) => candidate

case item: ItemCandidateWithDetails =>

throw PipelineFailure(

UnexpectedCandidateResult,

s"Invalid Item Candidate type expected $tag for Item Candidate type ${item.candidate.getClass}")

case \_: ModuleCandidateWithDetails =>

throw PipelineFailure(

UnexpectedCandidateResult,

"Cannot retrieve Item Candidate for a Module")

}

/\*\*

\* Convenience method for checking if this contains a certain candidate type

\*

\* @throws PipelineFailure if CandidateType does not match the expected Item Candidate type, or

\* if invoked on a Module Candidate

\*/

def isCandidateType[CandidateType <: UniversalNoun[\_]](

)(

implicit tag: ClassTag[CandidateType]

): Boolean = self match {

case ItemCandidateWithDetails(\_: CandidateType, \_, \_) => true

case \_ => false

}

}

case class ItemCandidateWithDetails(

override val candidate: UniversalNoun[Any],

presentation: Option[UniversalPresentation],

override val features: FeatureMap)

extends CandidateWithDetails

with CandidateWithFeatures[UniversalNoun[Any]]

case class ModuleCandidateWithDetails(

candidates: Seq[ItemCandidateWithDetails],

presentation: Option[ModulePresentation],

override val features: FeatureMap)

extends CandidateWithDetails

object ItemCandidateWithDetails {

def apply(

candidate: UniversalNoun[Any],

presentation: Option[UniversalPresentation],

source: CandidatePipelineIdentifier,

sourcePosition: Int,

features: FeatureMap

): ItemCandidateWithDetails = {

val newFeatureMap =

FeatureMapBuilder()

.add(CandidateSourcePosition, sourcePosition)

.add(CandidatePipelines, ListSet.empty + source).build() ++ features

ItemCandidateWithDetails(candidate, presentation, newFeatureMap)

}

}

object ModuleCandidateWithDetails {

def apply(

candidates: Seq[ItemCandidateWithDetails],

presentation: Option[ModulePresentation],

source: CandidatePipelineIdentifier,

sourcePosition: Int,

features: FeatureMap

): ModuleCandidateWithDetails = {

val newFeatureMap =

FeatureMapBuilder()

.add(CandidateSourcePosition, sourcePosition)

.add(CandidatePipelines, ListSet.empty + source).build() ++ features

ModuleCandidateWithDetails(candidates, presentation, newFeatureMap)

}

}