package com.twitter.product\_mixer.component\_library.decorator.urt

import com.twitter.product\_mixer.component\_library.model.presentation.urt.UrtItemPresentation

import com.twitter.product\_mixer.component\_library.model.presentation.urt.UrtModulePresentation

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

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

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

import com.twitter.product\_mixer.core.functional\_component.decorator.CandidateDecorator

import com.twitter.product\_mixer.core.functional\_component.decorator.Decoration

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

import com.twitter.stitch.Stitch

import com.twitter.product\_mixer.component\_library.decorator.urt.builder.timeline\_module.ModuleIdGeneration

import com.twitter.product\_mixer.component\_library.decorator.urt.builder.timeline\_module.AutomaticUniqueModuleId

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

import com.twitter.product\_mixer.core.functional\_component.decorator.urt.builder.timeline\_module.BaseTimelineModuleBuilder

/\*\*

\* Given a [[CandidateWithFeatures]] return the corresponding group with which it should be

\* associated. Returning none will result in the candidate not being assigned to any module.

\*/

trait GroupByKey[-Query <: PipelineQuery, -BuilderInput <: UniversalNoun[Any], Key] {

def apply(query: Query, candidate: BuilderInput, candidateFeatures: FeatureMap): Option[Key]

}

/\*\*

\* Similar to [[UrtItemInModuleDecorator]] except that this decorator can assign items to different

\* modules based on the provided [[GroupByKey]].

\*

\* @param urtItemCandidateDecorator decorates individual item candidates

\* @param moduleBuilder builds a module from a particular candidate group

\* @param groupByKey assigns each candidate a module group. Returning [[None]] will result in the

\* candidate not being assigned to a module

\*/

case class UrtMultipleModulesDecorator[

-Query <: PipelineQuery,

-BuilderInput <: UniversalNoun[Any],

GroupKey

](

urtItemCandidateDecorator: CandidateDecorator[Query, BuilderInput],

moduleBuilder: BaseTimelineModuleBuilder[Query, BuilderInput],

groupByKey: GroupByKey[Query, BuilderInput, GroupKey],

override val identifier: DecoratorIdentifier = DecoratorIdentifier("UrtMultipleModules"))

extends CandidateDecorator[Query, BuilderInput] {

override def apply(

query: Query,

candidates: Seq[CandidateWithFeatures[BuilderInput]]

): Stitch[Seq[Decoration]] = {

if (candidates.nonEmpty) {

/\*\* Individual candidates with [[UrtItemPresentation]]s \*/

val decoratedCandidatesStitch: Stitch[

Seq[(CandidateWithFeatures[BuilderInput], Decoration)]

] = urtItemCandidateDecorator(query, candidates).map(candidates.zip(\_))

decoratedCandidatesStitch.map { decoratedCandidates =>

// Group candidates into modules

val candidatesByModule: Map[Option[GroupKey], Seq[

(CandidateWithFeatures[BuilderInput], Decoration)

]] =

decoratedCandidates.groupBy {

case (CandidateWithFeatures(candidate, features), \_) =>

groupByKey(query, candidate, features)

}

candidatesByModule.iterator.zipWithIndex.flatMap {

// A None group key indicates these candidates should not be put into a module. Return

// the decorated candidates.

case ((None, candidateGroup), \_) =>

candidateGroup.map {

case (\_, decoration) => decoration

}

// Build a UrtModulePresentation and add it to each candidate's decoration.

case ((\_, candidateGroup), index) =>

val (candidatesWithFeatures, decorations) = candidateGroup.unzip

/\*\*

\* Build the module and update its ID if [[AutomaticUniqueModuleId]]s are being used.

\* Forcing IDs to be different ensures that modules are never accidentally grouped

\* together, since all other fields might otherwise be equal (candidates aren't added

\* to modules until the domain marshalling phase).

\*/

val timelineModule = {

val module = moduleBuilder(query, candidatesWithFeatures)

ModuleIdGeneration(module.id) match {

case id: AutomaticUniqueModuleId => module.copy(id = id.withOffset(index).moduleId)

case \_ => module

}

}

val modulePresentation = UrtModulePresentation(timelineModule)

decorations.collect {

case Decoration(candidate, urtItemPresentation: UrtItemPresentation) =>

Decoration(

candidate,

urtItemPresentation.copy(modulePresentation = Some(modulePresentation)))

}

}.toSeq

}

} else {

Stitch.Nil

}

}

}