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.model.marshalling.response.urt.TimelineItem

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.functional\_component.decorator.urt.builder.timeline\_module.BaseTimelineModuleBuilder

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

import com.twitter.stitch.Stitch

/\*\*

\* Decorator that will apply the provided [[urtItemCandidateDecorator]] to all the `candidates` and apply

\* the same [[UrtModulePresentation]] from [[moduleBuilder]] to each Candidate.

\*/

case class UrtItemInModuleDecorator[

Query <: PipelineQuery,

BuilderInput <: UniversalNoun[Any],

BuilderOutput <: TimelineItem

](

urtItemCandidateDecorator: CandidateDecorator[Query, BuilderInput],

moduleBuilder: BaseTimelineModuleBuilder[Query, BuilderInput],

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

extends CandidateDecorator[Query, BuilderInput] {

override def apply(

query: Query,

candidates: Seq[CandidateWithFeatures[BuilderInput]]

): Stitch[Seq[Decoration]] = {

if (candidates.nonEmpty) {

val urtItemCandidatesWithDecoration = urtItemCandidateDecorator(query, candidates)

// Pass candidates to support when the module is constructed dynamically based on the list

val modulePresentation =

UrtModulePresentation(moduleBuilder(query, candidates))

urtItemCandidatesWithDecoration.map { candidates =>

candidates.collect {

case Decoration(candidate, urtItemPresentation: UrtItemPresentation) =>

Decoration(

candidate,

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

}

}

} else {

Stitch.Nil

}

}

}