package com.twitter.home\_mixer.functional\_component.selector

import com.twitter.home\_mixer.functional\_component.decorator.builder.HomeClientEventDetailsBuilder

import com.twitter.home\_mixer.model.HomeFeatures.AncestorsFeature

import com.twitter.home\_mixer.model.HomeFeatures.ConversationModule2DisplayedTweetsFeature

import com.twitter.home\_mixer.model.HomeFeatures.ConversationModuleHasGapFeature

import com.twitter.home\_mixer.model.HomeFeatures.HasRandomTweetFeature

import com.twitter.home\_mixer.model.HomeFeatures.IsRandomTweetAboveFeature

import com.twitter.home\_mixer.model.HomeFeatures.IsRandomTweetFeature

import com.twitter.home\_mixer.model.HomeFeatures.PositionFeature

import com.twitter.home\_mixer.model.HomeFeatures.ServedInConversationModuleFeature

import com.twitter.home\_mixer.model.HomeFeatures.ServedSizeFeature

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.feature.featuremap.FeatureMap

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

import com.twitter.product\_mixer.core.functional\_component.common.CandidateScope

import com.twitter.product\_mixer.core.functional\_component.common.SpecificPipelines

import com.twitter.product\_mixer.core.functional\_component.selector.Selector

import com.twitter.product\_mixer.core.functional\_component.selector.SelectorResult

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

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

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

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

import com.twitter.product\_mixer.core.model.marshalling.response.urt.item.tweet.TweetItem

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

/\*\*

\* Builds serialized tweet type metrics controller data and updates Client Event Details

\* and Candidate Presentations with this info.

\*

\* Currently only updates presentation of Item Candidates. This needs to be updated

\* when modules are added.

\*

\* This is implemented as a Selector instead of a Decorator in the Candidate Pipeline

\* because we need to add controller data that looks at the final timeline as a whole

\* (e.g. served size, final candidate positions).

\*

\* @param candidatePipelines - only candidates from the specified pipeline will be updated

\*/

case class UpdateHomeClientEventDetails(candidatePipelines: Set[CandidatePipelineIdentifier])

extends Selector[PipelineQuery] {

override val pipelineScope: CandidateScope = SpecificPipelines(candidatePipelines)

private val detailsBuilder = HomeClientEventDetailsBuilder()

override def apply(

query: PipelineQuery,

remainingCandidates: Seq[CandidateWithDetails],

result: Seq[CandidateWithDetails]

): SelectorResult = {

val selectedCandidates = result.filter(pipelineScope.contains)

val randomTweetsByPosition = result

.map(\_.features.getOrElse(IsRandomTweetFeature, false))

.zipWithIndex.map(\_.swap).toMap

val resultFeatures = FeatureMapBuilder()

.add(ServedSizeFeature, Some(selectedCandidates.size))

.add(HasRandomTweetFeature, randomTweetsByPosition.valuesIterator.contains(true))

.build()

val updatedResult = result.zipWithIndex.map {

case (item @ ItemCandidateWithDetails(candidate, \_, \_), position)

if pipelineScope.contains(item) =>

val resultCandidateFeatures = FeatureMapBuilder()

.add(PositionFeature, Some(position))

.add(IsRandomTweetAboveFeature, randomTweetsByPosition.getOrElse(position - 1, false))

.build()

updateItemPresentation(query, item, resultFeatures, resultCandidateFeatures)

case (module @ ModuleCandidateWithDetails(candidates, presentation, features), position)

if pipelineScope.contains(module) =>

val resultCandidateFeatures = FeatureMapBuilder()

.add(PositionFeature, Some(position))

.add(IsRandomTweetAboveFeature, randomTweetsByPosition.getOrElse(position - 1, false))

.add(ServedInConversationModuleFeature, true)

.add(ConversationModule2DisplayedTweetsFeature, module.candidates.size == 2)

.add(

ConversationModuleHasGapFeature,

module.candidates.last.features.getOrElse(AncestorsFeature, Seq.empty).size > 2)

.build()

val updatedItemCandidates =

candidates.map(updateItemPresentation(query, \_, resultFeatures, resultCandidateFeatures))

val updatedCandidateFeatures = features ++ resultFeatures ++ resultCandidateFeatures

val updatedPresentation = presentation.map {

case urtModule @ UrtModulePresentation(timelineModule) =>

val clientEventDetails =

detailsBuilder(

query,

candidates.last.candidate,

query.features.get ++ updatedCandidateFeatures)

val updatedClientEventInfo =

timelineModule.clientEventInfo.map(\_.copy(details = clientEventDetails))

val updatedTimelineModule =

timelineModule.copy(clientEventInfo = updatedClientEventInfo)

urtModule.copy(timelineModule = updatedTimelineModule)

}

module.copy(

candidates = updatedItemCandidates,

presentation = updatedPresentation,

features = updatedCandidateFeatures

)

case (any, position) => any

}

SelectorResult(remainingCandidates = remainingCandidates, result = updatedResult)

}

private def updateItemPresentation(

query: PipelineQuery,

item: ItemCandidateWithDetails,

resultCandidateFeatures: FeatureMap,

resultFeatures: FeatureMap,

): ItemCandidateWithDetails = {

val updatedItemCandidateFeatures = item.features ++ resultFeatures ++ resultCandidateFeatures

val updatedPresentation = item.presentation.map {

case urtItem @ UrtItemPresentation(timelineItem: TweetItem, \_) =>

val clientEventDetails =

detailsBuilder(query, item.candidate, query.features.get ++ updatedItemCandidateFeatures)

val updatedClientEventInfo =

timelineItem.clientEventInfo.map(\_.copy(details = clientEventDetails))

val updatedTimelineItem = timelineItem.copy(clientEventInfo = updatedClientEventInfo)

urtItem.copy(timelineItem = updatedTimelineItem)

case any => any

}

item.copy(presentation = updatedPresentation, features = updatedItemCandidateFeatures)

}

}