package com.twitter.home\_mixer.functional\_component.side\_effect

import com.twitter.finagle.tracing.Trace

import com.twitter.home\_mixer.marshaller.timeline\_logging.PromotedTweetDetailsMarshaller

import com.twitter.home\_mixer.marshaller.timeline\_logging.TweetDetailsMarshaller

import com.twitter.home\_mixer.marshaller.timeline\_logging.WhoToFollowDetailsMarshaller

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

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

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

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

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

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

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

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

import com.twitter.home\_mixer.model.request.DeviceContext.RequestContext

import com.twitter.home\_mixer.model.request.HasDeviceContext

import com.twitter.home\_mixer.model.request.HasSeenTweetIds

import com.twitter.home\_mixer.model.request.FollowingProduct

import com.twitter.home\_mixer.model.request.ForYouProduct

import com.twitter.home\_mixer.model.request.SubscribedProduct

import com.twitter.home\_mixer.param.HomeMixerFlagName.ScribeServedCandidatesFlag

import com.twitter.home\_mixer.param.HomeGlobalParams.EnableScribeServedCandidatesParam

import com.twitter.home\_mixer.service.HomeMixerAlertConfig

import com.twitter.inject.annotations.Flag

import com.twitter.logpipeline.client.common.EventPublisher

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

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

import com.twitter.product\_mixer.component\_library.pipeline.candidate.who\_to\_follow\_module.WhoToFollowCandidateDecorator

import com.twitter.product\_mixer.component\_library.pipeline.candidate.who\_to\_subscribe\_module.WhoToSubscribeCandidateDecorator

import com.twitter.product\_mixer.component\_library.side\_effect.ScribeLogEventSideEffect

import com.twitter.product\_mixer.core.functional\_component.side\_effect.PipelineResultSideEffect

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

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.AddEntriesTimelineInstruction

import com.twitter.product\_mixer.core.model.marshalling.response.urt.ModuleItem

import com.twitter.product\_mixer.core.model.marshalling.response.urt.Timeline

import com.twitter.product\_mixer.core.model.marshalling.response.urt.TimelineModule

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

import com.twitter.product\_mixer.core.model.marshalling.response.urt.item.user.UserItem

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

import com.twitter.timelines.timeline\_logging.{thriftscala => thrift}

import com.twitter.util.Time

import javax.inject.Inject

import javax.inject.Singleton

/\*\*

\* Side effect that logs home timeline served candidates to Scribe.

\*/

@Singleton

class HomeScribeServedCandidatesSideEffect @Inject() (

@Flag(ScribeServedCandidatesFlag) enableScribeServedCandidates: Boolean,

scribeEventPublisher: EventPublisher[thrift.ServedEntry])

extends ScribeLogEventSideEffect[

thrift.ServedEntry,

PipelineQuery with HasSeenTweetIds with HasDeviceContext,

Timeline

]

with PipelineResultSideEffect.Conditionally[

PipelineQuery with HasSeenTweetIds with HasDeviceContext,

Timeline

] {

override val identifier: SideEffectIdentifier = SideEffectIdentifier("HomeScribeServedCandidates")

override def onlyIf(

query: PipelineQuery with HasSeenTweetIds with HasDeviceContext,

selectedCandidates: Seq[CandidateWithDetails],

remainingCandidates: Seq[CandidateWithDetails],

droppedCandidates: Seq[CandidateWithDetails],

response: Timeline

): Boolean = enableScribeServedCandidates && query.params(EnableScribeServedCandidatesParam)

override def buildLogEvents(

query: PipelineQuery with HasSeenTweetIds with HasDeviceContext,

selectedCandidates: Seq[CandidateWithDetails],

remainingCandidates: Seq[CandidateWithDetails],

droppedCandidates: Seq[CandidateWithDetails],

response: Timeline

): Seq[thrift.ServedEntry] = {

val timelineType = query.product match {

case FollowingProduct => thrift.TimelineType.HomeLatest

case ForYouProduct => thrift.TimelineType.Home

case SubscribedProduct => thrift.TimelineType.HomeSubscribed

case other => throw new UnsupportedOperationException(s"Unknown product: $other")

}

val requestProvenance = query.deviceContext.map { deviceContext =>

deviceContext.requestContextValue match {

case RequestContext.Foreground => thrift.RequestProvenance.Foreground

case RequestContext.Launch => thrift.RequestProvenance.Launch

case RequestContext.PullToRefresh => thrift.RequestProvenance.Ptr

case \_ => thrift.RequestProvenance.Other

}

}

val queryType = query.features.map { featureMap =>

if (featureMap.getOrElse(GetOlderFeature, false)) thrift.QueryType.GetOlder

else if (featureMap.getOrElse(GetNewerFeature, false)) thrift.QueryType.GetNewer

else if (featureMap.getOrElse(GetMiddleFeature, false)) thrift.QueryType.GetMiddle

else if (featureMap.getOrElse(GetInitialFeature, false)) thrift.QueryType.GetInitial

else thrift.QueryType.Other

}

val requestInfo = thrift.RequestInfo(

requestTimeMs = query.queryTime.inMilliseconds,

traceId = Trace.id.traceId.toLong,

userId = query.getOptionalUserId,

clientAppId = query.clientContext.appId,

hasDarkRequest = query.features.flatMap(\_.getOrElse(HasDarkRequestFeature, None)),

parentId = Some(Trace.id.parentId.toLong),

spanId = Some(Trace.id.spanId.toLong),

timelineType = Some(timelineType),

ipAddress = query.clientContext.ipAddress,

userAgent = query.clientContext.userAgent,

queryType = queryType,

requestProvenance = requestProvenance,

languageCode = query.clientContext.languageCode,

countryCode = query.clientContext.countryCode,

requestEndTimeMs = Some(Time.now.inMilliseconds),

servedRequestId = query.features.flatMap(\_.getOrElse(ServedRequestIdFeature, None)),

requestJoinId = query.features.flatMap(\_.getOrElse(RequestJoinIdFeature, None))

)

val tweetIdToItemCandidateMap: Map[Long, ItemCandidateWithDetails] =

selectedCandidates.flatMap {

case item: ItemCandidateWithDetails if item.candidate.isInstanceOf[BaseTweetCandidate] =>

Seq((item.candidateIdLong, item))

case module: ModuleCandidateWithDetails

if module.candidates.headOption.exists(\_.candidate.isInstanceOf[BaseTweetCandidate]) =>

module.candidates.map(item => (item.candidateIdLong, item))

case \_ => Seq.empty

}.toMap

val userIdToItemCandidateMap: Map[Long, ItemCandidateWithDetails] =

selectedCandidates.flatMap {

case module: ModuleCandidateWithDetails

if module.candidates.forall(\_.candidate.isInstanceOf[BaseUserCandidate]) =>

module.candidates.map { item =>

(item.candidateIdLong, item)

}

case \_ => Seq.empty

}.toMap

response.instructions.zipWithIndex

.collect {

case (AddEntriesTimelineInstruction(entries), index) =>

entries.collect {

case entry: TweetItem if entry.promotedMetadata.isDefined =>

val promotedTweetDetails = PromotedTweetDetailsMarshaller(entry, index)

Seq(

thrift.EntryInfo(

id = entry.id,

position = index.shortValue(),

entryId = entry.entryIdentifier,

entryType = thrift.EntryType.PromotedTweet,

sortIndex = entry.sortIndex,

verticalSize = Some(1),

displayType = Some(entry.displayType.toString),

details = Some(thrift.ItemDetails.PromotedTweetDetails(promotedTweetDetails))

)

)

case entry: TweetItem =>

val candidate = tweetIdToItemCandidateMap(entry.id)

val tweetDetails = TweetDetailsMarshaller(entry, candidate)

Seq(

thrift.EntryInfo(

id = candidate.candidateIdLong,

position = index.shortValue(),

entryId = entry.entryIdentifier,

entryType = thrift.EntryType.Tweet,

sortIndex = entry.sortIndex,

verticalSize = Some(1),

score = candidate.features.getOrElse(ScoreFeature, None),

displayType = Some(entry.displayType.toString),

details = Some(thrift.ItemDetails.TweetDetails(tweetDetails))

)

)

case module: TimelineModule

if module.entryNamespace.toString == WhoToFollowCandidateDecorator.EntryNamespaceString =>

module.items.collect {

case ModuleItem(entry: UserItem, \_, \_) =>

val candidate = userIdToItemCandidateMap(entry.id)

val whoToFollowDetails = WhoToFollowDetailsMarshaller(entry, candidate)

thrift.EntryInfo(

id = entry.id,

position = index.shortValue(),

entryId = module.entryIdentifier,

entryType = thrift.EntryType.WhoToFollowModule,

sortIndex = module.sortIndex,

score = candidate.features.getOrElse(ScoreFeature, None),

displayType = Some(entry.displayType.toString),

details = Some(thrift.ItemDetails.WhoToFollowDetails(whoToFollowDetails))

)

}

case module: TimelineModule

if module.entryNamespace.toString == WhoToSubscribeCandidateDecorator.EntryNamespaceString =>

module.items.collect {

case ModuleItem(entry: UserItem, \_, \_) =>

val candidate = userIdToItemCandidateMap(entry.id)

val whoToSubscribeDetails = WhoToFollowDetailsMarshaller(entry, candidate)

thrift.EntryInfo(

id = entry.id,

position = index.shortValue(),

entryId = module.entryIdentifier,

entryType = thrift.EntryType.WhoToSubscribeModule,

sortIndex = module.sortIndex,

score = candidate.features.getOrElse(ScoreFeature, None),

displayType = Some(entry.displayType.toString),

details = Some(thrift.ItemDetails.WhoToFollowDetails(whoToSubscribeDetails))

)

}

case module: TimelineModule

if module.sortIndex.isDefined && module.items.headOption.exists(

\_.item.isInstanceOf[TweetItem]) =>

module.items.collect {

case ModuleItem(entry: TweetItem, \_, \_) =>

val candidate = tweetIdToItemCandidateMap(entry.id)

thrift.EntryInfo(

id = entry.id,

position = index.shortValue(),

entryId = module.entryIdentifier,

entryType = thrift.EntryType.ConversationModule,

sortIndex = module.sortIndex,

score = candidate.features.getOrElse(ScoreFeature, None),

displayType = Some(entry.displayType.toString)

)

}

case \_ => Seq.empty

}.flatten

// Other instructions

case \_ => Seq.empty[thrift.EntryInfo]

}.flatten.map { entryInfo =>

thrift.ServedEntry(

entry = Some(entryInfo),

request = requestInfo

)

}

}

override val logPipelinePublisher: EventPublisher[thrift.ServedEntry] =

scribeEventPublisher

override val alerts = Seq(

HomeMixerAlertConfig.BusinessHours.defaultSuccessRateAlert()

)

}