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

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

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

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

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

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

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.core.feature.featuremap.FeatureMap

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.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.ReplaceEntryTimelineInstruction

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

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.pipeline.PipelineQuery

import com.twitter.stitch.Stitch

import com.twitter.timelinemixer.clients.persistence.EntryWithItemIds

import com.twitter.timelinemixer.clients.persistence.ItemIds

import com.twitter.timelinemixer.clients.persistence.TimelineResponseBatchesClient

import com.twitter.timelinemixer.clients.persistence.TimelineResponseV3

import com.twitter.timelines.persistence.thriftscala.TweetScoreV1

import com.twitter.timelines.persistence.{thriftscala => persistence}

import com.twitter.timelineservice.model.TimelineQuery

import com.twitter.timelineservice.model.TimelineQueryOptions

import com.twitter.timelineservice.model.TweetScore

import com.twitter.timelineservice.model.core.TimelineKind

import com.twitter.timelineservice.model.rich.EntityIdType

import com.twitter.util.Time

import com.twitter.{timelineservice => tls}

import javax.inject.Inject

import javax.inject.Singleton

object UpdateTimelinesPersistenceStoreSideEffect {

val EmptyItemIds = ItemIds(

None,

None,

None,

None,

None,

None,

None,

None,

None,

None,

None,

None,

None,

None,

None)

}

/\*\*

\* Side effect that updates the Timelines Persistence Store (Manhattan) with the entries being returned.

\*/

@Singleton

class UpdateTimelinesPersistenceStoreSideEffect @Inject() (

timelineResponseBatchesClient: TimelineResponseBatchesClient[TimelineResponseV3])

extends PipelineResultSideEffect[PipelineQuery, Timeline] {

override val identifier: SideEffectIdentifier =

SideEffectIdentifier("UpdateTimelinesPersistenceStore")

final override def apply(

inputs: PipelineResultSideEffect.Inputs[PipelineQuery, Timeline]

): Stitch[Unit] = {

if (inputs.response.instructions.nonEmpty) {

val timelineKind = inputs.query.product match {

case FollowingProduct => TimelineKind.homeLatest

case ForYouProduct => TimelineKind.home

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

}

val timelineQuery = TimelineQuery(

id = inputs.query.getRequiredUserId,

kind = timelineKind,

options = TimelineQueryOptions(

contextualUserId = inputs.query.getOptionalUserId,

deviceContext = tls.DeviceContext.empty.copy(

userAgent = inputs.query.clientContext.userAgent,

clientAppId = inputs.query.clientContext.appId)

)

)

val tweetIdToItemCandidateMap: Map[Long, ItemCandidateWithDetails] =

inputs.selectedCandidates.flatMap {

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

Seq((item.candidateIdLong, item))

case module: ModuleCandidateWithDetails

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

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

case \_ => Seq.empty

}.toMap

val entries = inputs.response.instructions.collect {

case AddEntriesTimelineInstruction(entries) =>

entries.collect {

// includes tweets, tweet previews, and promoted tweets

case entry: TweetItem if entry.sortIndex.isDefined => {

Seq(

buildTweetEntryWithItemIds(

tweetIdToItemCandidateMap(entry.id),

entry.sortIndex.get

))

}

// tweet conversation modules are flattened to individual tweets in the persistence store

case module: TimelineModule

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

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

module.items.map { item =>

buildTweetEntryWithItemIds(

tweetIdToItemCandidateMap(item.item.id.asInstanceOf[Long]),

module.sortIndex.get)

}

case module: TimelineModule

if module.sortIndex.isDefined && module.entryNamespace.toString == WhoToFollowCandidateDecorator.EntryNamespaceString =>

val userIds = module.items

.map(item =>

UpdateTimelinesPersistenceStoreSideEffect.EmptyItemIds.copy(userId =

Some(item.item.id.asInstanceOf[Long])))

Seq(

EntryWithItemIds(

entityIdType = EntityIdType.WhoToFollow,

sortIndex = module.sortIndex.get,

size = module.items.size.toShort,

itemIds = Some(userIds)

))

case module: TimelineModule

if module.sortIndex.isDefined && module.entryNamespace.toString == WhoToSubscribeCandidateDecorator.EntryNamespaceString =>

val userIds = module.items

.map(item =>

UpdateTimelinesPersistenceStoreSideEffect.EmptyItemIds.copy(userId =

Some(item.item.id.asInstanceOf[Long])))

Seq(

EntryWithItemIds(

entityIdType = EntityIdType.WhoToSubscribe,

sortIndex = module.sortIndex.get,

size = module.items.size.toShort,

itemIds = Some(userIds)

))

}.flatten

case ShowCoverInstruction(cover) =>

Seq(

EntryWithItemIds(

entityIdType = EntityIdType.Prompt,

sortIndex = cover.sortIndex.get,

size = 1,

itemIds = None

)

)

case ReplaceEntryTimelineInstruction(entry) =>

val namespaceLength = TweetItem.TweetEntryNamespace.toString.length

Seq(

EntryWithItemIds(

entityIdType = EntityIdType.Tweet,

sortIndex = entry.sortIndex.get,

size = 1,

itemIds = Some(

Seq(

ItemIds(

tweetId =

entry.entryIdToReplace.map(e => e.substring(namespaceLength + 1).toLong),

sourceTweetId = None,

quoteTweetId = None,

sourceAuthorId = None,

quoteAuthorId = None,

inReplyToTweetId = None,

inReplyToAuthorId = None,

semanticCoreId = None,

articleId = None,

hasRelevancePrompt = None,

promptData = None,

tweetScore = None,

entryIdToReplace = entry.entryIdToReplace,

tweetReactiveData = None,

userId = None

)

))

)

)

}.flatten

val response = TimelineResponseV3(

clientPlatform = timelineQuery.clientPlatform,

servedTime = Time.now,

requestType = requestTypeFromQuery(inputs.query),

entries = entries)

Stitch.callFuture(timelineResponseBatchesClient.insertResponse(timelineQuery, response))

} else Stitch.Unit

}

override val alerts = Seq(

HomeMixerAlertConfig.BusinessHours.defaultSuccessRateAlert(99.8)

)

private def buildTweetEntryWithItemIds(

candidate: ItemCandidateWithDetails,

sortIndex: Long

): EntryWithItemIds = {

val features = candidate.features

val sourceAuthorId =

if (features.getOrElse(IsRetweetFeature, false)) features.getOrElse(SourceUserIdFeature, None)

else features.getOrElse(AuthorIdFeature, None)

val quoteAuthorId =

if (features.getOrElse(QuotedTweetIdFeature, None).nonEmpty)

features.getOrElse(SourceUserIdFeature, None)

else None

val tweetScore = features.getOrElse(ScoreFeature, None).map { score =>

TweetScore.fromThrift(persistence.TweetScore.TweetScoreV1(TweetScoreV1(score)))

}

val itemIds = ItemIds(

tweetId = Some(candidate.candidateIdLong),

sourceTweetId = features.getOrElse(SourceTweetIdFeature, None),

quoteTweetId = features.getOrElse(QuotedTweetIdFeature, None),

sourceAuthorId = sourceAuthorId,

quoteAuthorId = quoteAuthorId,

inReplyToTweetId = features.getOrElse(InReplyToTweetIdFeature, None),

inReplyToAuthorId = features.getOrElse(DirectedAtUserIdFeature, None),

semanticCoreId = features.getOrElse(SemanticCoreIdFeature, None),

articleId = None,

hasRelevancePrompt = None,

promptData = None,

tweetScore = tweetScore,

entryIdToReplace = None,

tweetReactiveData = None,

userId = None

)

val isPreview = features.getOrElse(IsTweetPreviewFeature, default = false)

val entityType = if (isPreview) EntityIdType.TweetPreview else EntityIdType.Tweet

EntryWithItemIds(

entityIdType = entityType,

sortIndex = sortIndex,

size = 1.toShort,

itemIds = Some(Seq(itemIds))

)

}

private def requestTypeFromQuery(query: PipelineQuery): persistence.RequestType = {

val features = query.features.getOrElse(FeatureMap.empty)

val featureToRequestType = Seq(

(PollingFeature, persistence.RequestType.Polling),

(GetInitialFeature, persistence.RequestType.Initial),

(GetNewerFeature, persistence.RequestType.Newer),

(GetMiddleFeature, persistence.RequestType.Middle),

(GetOlderFeature, persistence.RequestType.Older)

)

featureToRequestType

.collectFirst {

case (feature, requestType) if features.getOrElse(feature, false) => requestType

}.getOrElse(persistence.RequestType.Other)

}

}