package com.twitter.home\_mixer.product.for\_you.feature\_hydrator

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

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

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

import com.twitter.product\_mixer.core.feature.Feature

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.feature\_hydrator.BulkCandidateFeatureHydrator

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

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

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

import com.twitter.stitch.Stitch

import com.twitter.strato.generated.client.audiencerewards.audienceRewardsService.GetCreatorPreferencesOnUserClientColumn

import javax.inject.Inject

import javax.inject.Singleton

/\*\*

\* Hydrates the `AuthorEnabledPreviews` feature for tweets authored by creators by querying the

\* `GetCreatorPreferences` Strato column. This feature corresponds to the `previews\_enabled` field of that column.

\* Given a tweet from a creator, this feature indicates whether that creator has enabled previews

\* on their profile.

\*/

@Singleton

class AuthorEnabledPreviewsFeatureHydrator @Inject() (

getCreatorPreferencesOnUserClientColumn: GetCreatorPreferencesOnUserClientColumn)

extends BulkCandidateFeatureHydrator[PipelineQuery, TweetCandidate] {

override val identifier: FeatureHydratorIdentifier =

FeatureHydratorIdentifier("AuthorEnabledPreviews")

override val features: Set[Feature[\_, \_]] = Set(AuthorEnabledPreviewsFeature)

private val fetcher = getCreatorPreferencesOnUserClientColumn.fetcher

private val DefaultAuthorEnabledPreviewsValue = true

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[Seq[FeatureMap]] = {

val candidateAuthors = candidates

.map(\_.features.getOrElse(AuthorIdFeature, None))

.toSet

.flatten

// Build a map of creator -> authorEnabledPreviews, then use it to populate candidate features

val authorIdToFeatureStitch = Stitch.collect {

candidateAuthors

.map { author =>

val isAuthorEnabledPreviews = fetcher.fetch(author).map {

\_.v.map(\_.previewsEnabled).getOrElse(DefaultAuthorEnabledPreviewsValue)

}

(author, isAuthorEnabledPreviews)

}.toMap

}

authorIdToFeatureStitch.map { authorIdToFeatureMap =>

candidates.map {

\_.features.getOrElse(AuthorIdFeature, None) match {

case Some(authorId) => FeatureMapBuilder()

.add(AuthorEnabledPreviewsFeature, authorIdToFeatureMap(authorId))

.build()

case \_ => FeatureMapBuilder()

.add(AuthorEnabledPreviewsFeature, DefaultAuthorEnabledPreviewsValue)

.build()

}

}

}

}

}