package com.twitter.simclusters\_v2.stores

import com.twitter.simclusters\_v2.common.ClusterId

import com.twitter.simclusters\_v2.common.SimClustersEmbedding

import com.twitter.simclusters\_v2.thriftscala.ClusterDetails

import com.twitter.simclusters\_v2.thriftscala.InternalId

import com.twitter.simclusters\_v2.thriftscala.ModelVersion

import com.twitter.simclusters\_v2.thriftscala.SimClustersEmbeddingId

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Future

/\*\*

\* Transfer a Entity SimClustersEmbedding to a language filtered embedding.

\* The new embedding only contains clusters whose main language is the same as the language field in

\* the SimClustersEmbeddingId.

\*

\* This store is special designed for Topic Tweet and Topic Follow Prompt.

\* Only support new Ids whose internalId is LocaleEntityId.

\*/

@deprecated

case class LanguageFilteredLocaleEntityEmbeddingStore(

underlyingStore: ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding],

clusterDetailsStore: ReadableStore[(ModelVersion, ClusterId), ClusterDetails],

composeKeyMapping: SimClustersEmbeddingId => SimClustersEmbeddingId)

extends ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding] {

import LanguageFilteredLocaleEntityEmbeddingStore.\_

override def get(k: SimClustersEmbeddingId): Future[Option[SimClustersEmbedding]] = {

for {

maybeEmbedding <- underlyingStore.get(composeKeyMapping(k))

maybeFilteredEmbedding <- maybeEmbedding match {

case Some(embedding) =>

embeddingsLanguageFilter(k, embedding).map(Some(\_))

case None =>

Future.None

}

} yield maybeFilteredEmbedding

}

private def embeddingsLanguageFilter(

sourceEmbeddingId: SimClustersEmbeddingId,

simClustersEmbedding: SimClustersEmbedding

): Future[SimClustersEmbedding] = {

val language = getLanguage(sourceEmbeddingId)

val modelVersion = sourceEmbeddingId.modelVersion

val clusterDetailKeys = simClustersEmbedding.sortedClusterIds.map { clusterId =>

(modelVersion, clusterId)

}.toSet

Future

.collect {

clusterDetailsStore.multiGet(clusterDetailKeys)

}.map { clusterDetailsMap =>

simClustersEmbedding.embedding.filter {

case (clusterId, \_) =>

isDominantLanguage(

language,

clusterDetailsMap.getOrElse((modelVersion, clusterId), None))

}

}.map(SimClustersEmbedding(\_))

}

private def isDominantLanguage(

requestLang: String,

clusterDetails: Option[ClusterDetails]

): Boolean =

clusterDetails match {

case Some(details) =>

val dominantLanguage =

details.languageToFractionDeviceLanguage.map { langMap =>

langMap.maxBy {

case (\_, score) => score

}.\_1

}

dominantLanguage.exists(\_.equalsIgnoreCase(requestLang))

case \_ => true

}

}

object LanguageFilteredLocaleEntityEmbeddingStore {

def getLanguage(simClustersEmbeddingId: SimClustersEmbeddingId): String = {

simClustersEmbeddingId match {

case SimClustersEmbeddingId(\_, \_, InternalId.LocaleEntityId(localeEntityId)) =>

localeEntityId.language

case \_ =>

throw new IllegalArgumentException(

s"The Id $simClustersEmbeddingId doesn't contain Locale info")

}

}

}