package com.twitter.frigate.pushservice.store

import com.twitter.context.TwitterContext

import com.twitter.context.thriftscala.Viewer

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.TwitterContextPermit

import com.twitter.frigate.pushservice.model.PushTypes.Target

import com.twitter.frigate.pushservice.params.PushFeatureSwitchParams

import com.twitter.frigate.pushservice.params.PushParams

import com.twitter.frigate.thriftscala.CommonRecommendationType

import com.twitter.kujaku.domain.thriftscala.CacheUsageType

import com.twitter.kujaku.domain.thriftscala.MachineTranslation

import com.twitter.kujaku.domain.thriftscala.MachineTranslationResponse

import com.twitter.kujaku.domain.thriftscala.TranslationSource

import com.twitter.storehaus.ReadableStore

import com.twitter.strato.generated.client.translation.service.IsTweetTranslatableClientColumn

import com.twitter.strato.generated.client.translation.service.platform.MachineTranslateTweetClientColumn

import com.twitter.tweetypie.thriftscala.Tweet

import com.twitter.util.Future

import com.twitter.util.logging.Logging

object TweetTranslationStore {

case class Key(

target: Target,

tweetId: Long,

tweet: Option[Tweet],

crt: CommonRecommendationType)

case class Value(

translatedTweetText: String,

localizedSourceLanguage: String)

val allowedCRTs = Set[CommonRecommendationType](

CommonRecommendationType.TwistlyTweet

)

}

case class TweetTranslationStore(

translateTweetStore: ReadableStore[

MachineTranslateTweetClientColumn.Key,

MachineTranslationResponse

],

isTweetTranslatableStore: ReadableStore[IsTweetTranslatableClientColumn.Key, Boolean],

statsReceiver: StatsReceiver)

extends ReadableStore[TweetTranslationStore.Key, TweetTranslationStore.Value]

with Logging {

private val stats = statsReceiver.scope("tweetTranslationStore")

private val isTranslatableCounter = stats.counter("tweetIsTranslatable")

private val notTranslatableCounter = stats.counter("tweetIsNotTranslatable")

private val protectedUserCounter = stats.counter("protectedUser")

private val notProtectedUserCounter = stats.counter("notProtectedUser")

private val validLanguageCounter = stats.counter("validTweetLanguage")

private val invalidLanguageCounter = stats.counter("invalidTweetLanguage")

private val validCrtCounter = stats.counter("validCrt")

private val invalidCrtCounter = stats.counter("invalidCrt")

private val paramEnabledCounter = stats.counter("paramEnabled")

private val paramDisabledCounter = stats.counter("paramDisabled")

private val twitterContext = TwitterContext(TwitterContextPermit)

override def get(k: TweetTranslationStore.Key): Future[Option[TweetTranslationStore.Value]] = {

k.target.inferredUserDeviceLanguage.flatMap {

case Some(deviceLanguage) =>

setTwitterContext(k.target, deviceLanguage) {

translateTweet(

target = k.target,

tweetId = k.tweetId,

tweet = k.tweet,

crt = k.crt,

deviceLanguage = deviceLanguage).map { responseOpt =>

responseOpt.flatMap { response =>

response.translatorLocalizedSourceLanguage

.map { localizedSourceLanguage =>

TweetTranslationStore.Value(

translatedTweetText = response.translation,

localizedSourceLanguage = localizedSourceLanguage

)

}.filter { \_ =>

response.translationSource == TranslationSource.Google

}

}

}

}

case None => Future.None

}

}

// Don't sent protected tweets to external API for translation

private def checkProtectedUser(target: Target): Future[Boolean] = {

target.targetUser.map(\_.flatMap(\_.safety).forall(\_.isProtected)).onSuccess {

case true => protectedUserCounter.incr()

case false => notProtectedUserCounter.incr()

}

}

private def isTweetTranslatable(

target: Target,

tweetId: Long,

tweet: Option[Tweet],

crt: CommonRecommendationType,

deviceLanguage: String

): Future[Boolean] = {

val tweetLangOpt = tweet.flatMap(\_.language)

val isValidLanguage = tweetLangOpt.exists { tweetLang =>

tweetLang.confidence > 0.5 &&

tweetLang.language != deviceLanguage

}

if (isValidLanguage) {

validLanguageCounter.incr()

} else {

invalidLanguageCounter.incr()

}

val isValidCrt = TweetTranslationStore.allowedCRTs.contains(crt)

if (isValidCrt) {

validCrtCounter.incr()

} else {

invalidCrtCounter.incr()

}

if (isValidCrt && isValidLanguage && target.params(PushParams.EnableIsTweetTranslatableCheck)) {

checkProtectedUser(target).flatMap {

case false =>

val isTweetTranslatableKey = IsTweetTranslatableClientColumn.Key(

tweetId = tweetId,

destinationLanguage = Some(deviceLanguage),

translationSource = Some(TranslationSource.Google.name),

excludePreferredLanguages = Some(true)

)

isTweetTranslatableStore

.get(isTweetTranslatableKey).map { resultOpt =>

resultOpt.getOrElse(false)

}.onSuccess {

case true => isTranslatableCounter.incr()

case false => notTranslatableCounter.incr()

}

case true =>

Future.False

}

} else {

Future.False

}

}

private def translateTweet(

tweetId: Long,

deviceLanguage: String

): Future[Option[MachineTranslation]] = {

val translateKey = MachineTranslateTweetClientColumn.Key(

tweetId = tweetId,

destinationLanguage = deviceLanguage,

translationSource = TranslationSource.Google,

translatableEntityTypes = Seq(),

onlyCached = false,

cacheUsageType = CacheUsageType.Default

)

translateTweetStore.get(translateKey).map {

\_.collect {

case MachineTranslationResponse.Result(result) => result

}

}

}

private def translateTweet(

target: Target,

tweetId: Long,

tweet: Option[Tweet],

crt: CommonRecommendationType,

deviceLanguage: String

): Future[Option[MachineTranslation]] = {

isTweetTranslatable(target, tweetId, tweet, crt, deviceLanguage).flatMap {

case true =>

val isEnabledByParam = target.params(PushFeatureSwitchParams.EnableTweetTranslation)

if (isEnabledByParam) {

paramEnabledCounter.incr()

translateTweet(tweetId, deviceLanguage)

} else {

paramDisabledCounter.incr()

Future.None

}

case false =>

Future.None

}

}

private def setTwitterContext[Rep](

target: Target,

deviceLanguage: String

)(

f: => Future[Rep]

): Future[Rep] = {

twitterContext() match {

case Some(viewer) if viewer.userId.nonEmpty && viewer.authenticatedUserId.nonEmpty =>

// If the context is already setup with a user ID just use it

f

case \_ =>

// If not, create a new context containing the viewer user id

twitterContext.let(

Viewer(

userId = Some(target.targetId),

requestLanguageCode = Some(deviceLanguage),

authenticatedUserId = Some(target.targetId)

)) {

f

}

}

}

}