package com.twitter.tweetypie

package store

import com.fasterxml.jackson.databind.ObjectMapper

import com.fasterxml.jackson.module.scala.DefaultScalaModule

import com.twitter.finagle.tracing.Trace

import com.twitter.tweetypie.additionalfields.AdditionalFields

import com.twitter.tweetypie.client\_id.ClientIdHelper

import com.twitter.tweetypie.media.Media.ownMedia

trait LogLensStore

extends TweetStoreBase[LogLensStore]

with InsertTweet.Store

with DeleteTweet.Store

with UndeleteTweet.Store

with SetAdditionalFields.Store

with DeleteAdditionalFields.Store

with ScrubGeo.Store

with Takedown.Store

with UpdatePossiblySensitiveTweet.Store {

def wrap(w: TweetStore.Wrap): LogLensStore =

new TweetStoreWrapper(w, this)

with LogLensStore

with InsertTweet.StoreWrapper

with DeleteTweet.StoreWrapper

with UndeleteTweet.StoreWrapper

with SetAdditionalFields.StoreWrapper

with DeleteAdditionalFields.StoreWrapper

with ScrubGeo.StoreWrapper

with Takedown.StoreWrapper

with UpdatePossiblySensitiveTweet.StoreWrapper

}

object LogLensStore {

def apply(

tweetCreationsLogger: Logger,

tweetDeletionsLogger: Logger,

tweetUndeletionsLogger: Logger,

tweetUpdatesLogger: Logger,

clientIdHelper: ClientIdHelper,

): LogLensStore =

new LogLensStore {

private[this] val mapper = new ObjectMapper().registerModule(DefaultScalaModule)

private def logMessage(logger: Logger, data: (String, Any)\*): Future[Unit] =

Future {

val allData = data ++ defaultData

val msg = mapper.writeValueAsString(Map(allData: \_\*))

logger.info(msg)

}

// Note: Longs are logged as strings to avoid JSON 53-bit numeric truncation

private def defaultData: Seq[(String, Any)] = {

val viewer = TwitterContext()

Seq(

"client\_id" -> getOpt(clientIdHelper.effectiveClientId),

"service\_id" -> getOpt(clientIdHelper.effectiveServiceIdentifier),

"trace\_id" -> Trace.id.traceId.toString,

"audit\_ip" -> getOpt(viewer.flatMap(\_.auditIp)),

"application\_id" -> getOpt(viewer.flatMap(\_.clientApplicationId).map(\_.toString)),

"user\_agent" -> getOpt(viewer.flatMap(\_.userAgent)),

"authenticated\_user\_id" -> getOpt(viewer.flatMap(\_.authenticatedUserId).map(\_.toString))

)

}

private def getOpt[A](opt: Option[A]): Any =

opt.getOrElse(null)

override val insertTweet: FutureEffect[InsertTweet.Event] =

FutureEffect[InsertTweet.Event] { event =>

logMessage(

tweetCreationsLogger,

"type" -> "create\_tweet",

"tweet\_id" -> event.tweet.id.toString,

"user\_id" -> event.user.id.toString,

"source\_tweet\_id" -> getOpt(event.sourceTweet.map(\_.id.toString)),

"source\_user\_id" -> getOpt(event.sourceUser.map(\_.id.toString)),

"directed\_at\_user\_id" -> getOpt(getDirectedAtUser(event.tweet).map(\_.userId.toString)),

"reply\_to\_tweet\_id" -> getOpt(

getReply(event.tweet).flatMap(\_.inReplyToStatusId).map(\_.toString)),

"reply\_to\_user\_id" -> getOpt(getReply(event.tweet).map(\_.inReplyToUserId.toString)),

"media\_ids" -> ownMedia(event.tweet).map(\_.mediaId.toString)

)

}

override val deleteTweet: FutureEffect[DeleteTweet.Event] =

FutureEffect[DeleteTweet.Event] { event =>

logMessage(

tweetDeletionsLogger,

"type" -> "delete\_tweet",

"tweet\_id" -> event.tweet.id.toString,

"user\_id" -> getOpt(event.user.map(\_.id.toString)),

"source\_tweet\_id" -> getOpt(getShare(event.tweet).map(\_.sourceStatusId.toString)),

"by\_user\_id" -> getOpt(event.byUserId.map(\_.toString)),

"passthrough\_audit\_ip" -> getOpt(event.auditPassthrough.flatMap(\_.host)),

"media\_ids" -> ownMedia(event.tweet).map(\_.mediaId.toString),

"cascaded\_from\_tweet\_id" -> getOpt(event.cascadedFromTweetId.map(\_.toString))

)

}

override val undeleteTweet: FutureEffect[UndeleteTweet.Event] =

FutureEffect[UndeleteTweet.Event] { event =>

logMessage(

tweetUndeletionsLogger,

"type" -> "undelete\_tweet",

"tweet\_id" -> event.tweet.id.toString,

"user\_id" -> event.user.id.toString,

"source\_tweet\_id" -> getOpt(getShare(event.tweet).map(\_.sourceStatusId.toString)),

"media\_ids" -> ownMedia(event.tweet).map(\_.mediaId.toString)

)

}

override val setAdditionalFields: FutureEffect[SetAdditionalFields.Event] =

FutureEffect[SetAdditionalFields.Event] { event =>

logMessage(

tweetUpdatesLogger,

"type" -> "set\_additional\_fields",

"tweet\_id" -> event.additionalFields.id.toString,

"field\_ids" -> AdditionalFields.nonEmptyAdditionalFieldIds(event.additionalFields)

)

}

override val deleteAdditionalFields: FutureEffect[DeleteAdditionalFields.Event] =

FutureEffect[DeleteAdditionalFields.Event] { event =>

logMessage(

tweetUpdatesLogger,

"type" -> "delete\_additional\_fields",

"tweet\_id" -> event.tweetId.toString,

"field\_ids" -> event.fieldIds

)

}

override val scrubGeo: FutureEffect[ScrubGeo.Event] =

FutureEffect[ScrubGeo.Event] { event =>

Future.join(

event.tweetIds.map { tweetId =>

logMessage(

tweetUpdatesLogger,

"type" -> "scrub\_geo",

"tweet\_id" -> tweetId.toString,

"user\_id" -> event.userId.toString

)

}

)

}

override val takedown: FutureEffect[Takedown.Event] =

FutureEffect[Takedown.Event] { event =>

logMessage(

tweetUpdatesLogger,

"type" -> "takedown",

"tweet\_id" -> event.tweet.id.toString,

"user\_id" -> getUserId(event.tweet).toString,

"reasons" -> event.takedownReasons

)

}

override val updatePossiblySensitiveTweet: FutureEffect[UpdatePossiblySensitiveTweet.Event] =

FutureEffect[UpdatePossiblySensitiveTweet.Event] { event =>

logMessage(

tweetUpdatesLogger,

"type" -> "update\_possibly\_sensitive\_tweet",

"tweet\_id" -> event.tweet.id.toString,

"nsfw\_admin" -> TweetLenses.nsfwAdmin(event.tweet),

"nsfw\_user" -> TweetLenses.nsfwUser(event.tweet)

)

}

}

}