package com.twitter.tweetypie

package store

import com.twitter.tweetypie.thriftscala.\_

trait TweetEventBusStore

extends TweetStoreBase[TweetEventBusStore]

with AsyncDeleteAdditionalFields.Store

with AsyncDeleteTweet.Store

with AsyncInsertTweet.Store

with AsyncSetAdditionalFields.Store

with AsyncTakedown.Store

with AsyncUndeleteTweet.Store

with AsyncUpdatePossiblySensitiveTweet.Store

with QuotedTweetDelete.Store

with QuotedTweetTakedown.Store

with ScrubGeoUpdateUserTimestamp.Store

with ScrubGeo.Store { self =>

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

new TweetStoreWrapper(w, this)

with TweetEventBusStore

with AsyncDeleteAdditionalFields.StoreWrapper

with AsyncDeleteTweet.StoreWrapper

with AsyncInsertTweet.StoreWrapper

with AsyncSetAdditionalFields.StoreWrapper

with AsyncTakedown.StoreWrapper

with AsyncUndeleteTweet.StoreWrapper

with AsyncUpdatePossiblySensitiveTweet.StoreWrapper

with QuotedTweetDelete.StoreWrapper

with QuotedTweetTakedown.StoreWrapper

with ScrubGeo.StoreWrapper

with ScrubGeoUpdateUserTimestamp.StoreWrapper

def inParallel(that: TweetEventBusStore): TweetEventBusStore =

new TweetEventBusStore {

override val asyncInsertTweet: FutureEffect[AsyncInsertTweet.Event] =

self.asyncInsertTweet.inParallel(that.asyncInsertTweet)

override val asyncDeleteAdditionalFields: FutureEffect[AsyncDeleteAdditionalFields.Event] =

self.asyncDeleteAdditionalFields.inParallel(that.asyncDeleteAdditionalFields)

override val asyncDeleteTweet: FutureEffect[AsyncDeleteTweet.Event] =

self.asyncDeleteTweet.inParallel(that.asyncDeleteTweet)

override val asyncSetAdditionalFields: FutureEffect[AsyncSetAdditionalFields.Event] =

self.asyncSetAdditionalFields.inParallel(that.asyncSetAdditionalFields)

override val asyncTakedown: FutureEffect[AsyncTakedown.Event] =

self.asyncTakedown.inParallel(that.asyncTakedown)

override val asyncUndeleteTweet: FutureEffect[AsyncUndeleteTweet.Event] =

self.asyncUndeleteTweet.inParallel(that.asyncUndeleteTweet)

override val asyncUpdatePossiblySensitiveTweet: FutureEffect[

AsyncUpdatePossiblySensitiveTweet.Event

] =

self.asyncUpdatePossiblySensitiveTweet.inParallel(that.asyncUpdatePossiblySensitiveTweet)

override val quotedTweetDelete: FutureEffect[QuotedTweetDelete.Event] =

self.quotedTweetDelete.inParallel(that.quotedTweetDelete)

override val quotedTweetTakedown: FutureEffect[QuotedTweetTakedown.Event] =

self.quotedTweetTakedown.inParallel(that.quotedTweetTakedown)

override val retryAsyncInsertTweet: FutureEffect[

TweetStoreRetryEvent[AsyncInsertTweet.Event]

] =

self.retryAsyncInsertTweet.inParallel(that.retryAsyncInsertTweet)

override val retryAsyncDeleteAdditionalFields: FutureEffect[

TweetStoreRetryEvent[AsyncDeleteAdditionalFields.Event]

] =

self.retryAsyncDeleteAdditionalFields.inParallel(that.retryAsyncDeleteAdditionalFields)

override val retryAsyncDeleteTweet: FutureEffect[

TweetStoreRetryEvent[AsyncDeleteTweet.Event]

] =

self.retryAsyncDeleteTweet.inParallel(that.retryAsyncDeleteTweet)

override val retryAsyncUndeleteTweet: FutureEffect[

TweetStoreRetryEvent[AsyncUndeleteTweet.Event]

] =

self.retryAsyncUndeleteTweet.inParallel(that.retryAsyncUndeleteTweet)

override val retryAsyncUpdatePossiblySensitiveTweet: FutureEffect[

TweetStoreRetryEvent[AsyncUpdatePossiblySensitiveTweet.Event]

] =

self.retryAsyncUpdatePossiblySensitiveTweet.inParallel(

that.retryAsyncUpdatePossiblySensitiveTweet

)

override val retryAsyncSetAdditionalFields: FutureEffect[

TweetStoreRetryEvent[AsyncSetAdditionalFields.Event]

] =

self.retryAsyncSetAdditionalFields.inParallel(that.retryAsyncSetAdditionalFields)

override val retryAsyncTakedown: FutureEffect[TweetStoreRetryEvent[AsyncTakedown.Event]] =

self.retryAsyncTakedown.inParallel(that.retryAsyncTakedown)

override val scrubGeo: FutureEffect[ScrubGeo.Event] =

self.scrubGeo.inParallel(that.scrubGeo)

override val scrubGeoUpdateUserTimestamp: FutureEffect[ScrubGeoUpdateUserTimestamp.Event] =

self.scrubGeoUpdateUserTimestamp.inParallel(that.scrubGeoUpdateUserTimestamp)

}

}

object TweetEventBusStore {

val Action: AsyncWriteAction = AsyncWriteAction.EventBusEnqueue

def safetyTypeForUser(user: User): Option[SafetyType] =

user.safety.map(userSafetyToSafetyType)

def userSafetyToSafetyType(safety: Safety): SafetyType =

if (safety.isProtected) {

SafetyType.Private

} else if (safety.suspended) {

SafetyType.Restricted

} else {

SafetyType.Public

}

def apply(

eventStore: FutureEffect[TweetEvent]

): TweetEventBusStore = {

def toTweetEvents(event: TweetStoreTweetEvent): Seq[TweetEvent] =

event.toTweetEventData.map { data =>

TweetEvent(

data,

TweetEventFlags(

timestampMs = event.timestamp.inMillis,

safetyType = event.optUser.flatMap(safetyTypeForUser)

)

)

}

def enqueueEvents[E <: TweetStoreTweetEvent]: FutureEffect[E] =

eventStore.liftSeq.contramap[E](toTweetEvents)

new TweetEventBusStore {

override val asyncInsertTweet: FutureEffect[AsyncInsertTweet.Event] =

enqueueEvents[AsyncInsertTweet.Event]

override val asyncDeleteAdditionalFields: FutureEffect[AsyncDeleteAdditionalFields.Event] =

enqueueEvents[AsyncDeleteAdditionalFields.Event]

override val asyncDeleteTweet: FutureEffect[AsyncDeleteTweet.Event] =

enqueueEvents[AsyncDeleteTweet.Event]

override val asyncSetAdditionalFields: FutureEffect[AsyncSetAdditionalFields.Event] =

enqueueEvents[AsyncSetAdditionalFields.Event]

override val asyncTakedown: FutureEffect[AsyncTakedown.Event] =

enqueueEvents[AsyncTakedown.Event]

.onlyIf(\_.eventbusEnqueue)

override val asyncUndeleteTweet: FutureEffect[AsyncUndeleteTweet.Event] =

enqueueEvents[AsyncUndeleteTweet.Event]

override val asyncUpdatePossiblySensitiveTweet: FutureEffect[

AsyncUpdatePossiblySensitiveTweet.Event

] =

enqueueEvents[AsyncUpdatePossiblySensitiveTweet.Event]

override val quotedTweetDelete: FutureEffect[QuotedTweetDelete.Event] =

enqueueEvents[QuotedTweetDelete.Event]

override val quotedTweetTakedown: FutureEffect[QuotedTweetTakedown.Event] =

enqueueEvents[QuotedTweetTakedown.Event]

override val retryAsyncInsertTweet: FutureEffect[

TweetStoreRetryEvent[AsyncInsertTweet.Event]

] =

TweetStore.retry(Action, asyncInsertTweet)

override val retryAsyncDeleteAdditionalFields: FutureEffect[

TweetStoreRetryEvent[AsyncDeleteAdditionalFields.Event]

] =

TweetStore.retry(Action, asyncDeleteAdditionalFields)

override val retryAsyncDeleteTweet: FutureEffect[

TweetStoreRetryEvent[AsyncDeleteTweet.Event]

] =

TweetStore.retry(Action, asyncDeleteTweet)

override val retryAsyncUndeleteTweet: FutureEffect[

TweetStoreRetryEvent[AsyncUndeleteTweet.Event]

] =

TweetStore.retry(Action, asyncUndeleteTweet)

override val retryAsyncUpdatePossiblySensitiveTweet: FutureEffect[

TweetStoreRetryEvent[AsyncUpdatePossiblySensitiveTweet.Event]

] =

TweetStore.retry(Action, asyncUpdatePossiblySensitiveTweet)

override val retryAsyncSetAdditionalFields: FutureEffect[

TweetStoreRetryEvent[AsyncSetAdditionalFields.Event]

] =

TweetStore.retry(Action, asyncSetAdditionalFields)

override val retryAsyncTakedown: FutureEffect[TweetStoreRetryEvent[AsyncTakedown.Event]] =

TweetStore.retry(Action, asyncTakedown)

override val scrubGeo: FutureEffect[ScrubGeo.Event] =

enqueueEvents[ScrubGeo.Event]

override val scrubGeoUpdateUserTimestamp: FutureEffect[ScrubGeoUpdateUserTimestamp.Event] =

enqueueEvents[ScrubGeoUpdateUserTimestamp.Event]

}

}

}

/\*\*

\* Scrubs inappropriate fields from tweet events before publishing.

\*/

object TweetEventDataScrubber {

def scrub(tweet: Tweet): Tweet =

tweet.copy(

cards = None,

card2 = None,

media = tweet.media.map(\_.map { mediaEntity => mediaEntity.copy(extensionsReply = None) }),

previousCounts = None,

editPerspective = None

)

}