package com.twitter.tweetypie

package store

import com.twitter.tweetypie.thriftscala.\_

object SetAdditionalFields extends TweetStore.SyncModule {

case class Event(additionalFields: Tweet, userId: UserId, timestamp: Time)

extends SyncTweetStoreEvent("set\_additional\_fields") {

def toAsyncRequest: AsyncSetAdditionalFieldsRequest =

AsyncSetAdditionalFieldsRequest(

additionalFields = additionalFields,

userId = userId,

timestamp = timestamp.inMillis

)

}

trait Store {

val setAdditionalFields: FutureEffect[Event]

}

trait StoreWrapper extends Store { self: TweetStoreWrapper[Store] =>

override val setAdditionalFields: FutureEffect[Event] = wrap(underlying.setAdditionalFields)

}

object Store {

def apply(

manhattanStore: ManhattanTweetStore,

cachingTweetStore: CachingTweetStore,

asyncEnqueueStore: AsyncEnqueueStore,

logLensStore: LogLensStore

): Store =

new Store {

override val setAdditionalFields: FutureEffect[Event] =

FutureEffect.sequentially(

logLensStore.setAdditionalFields,

manhattanStore.setAdditionalFields,

// Ignore failures but wait for completion to ensure we attempted to update cache before

// running async tasks, in particular publishing an event to EventBus.

cachingTweetStore.ignoreFailuresUponCompletion.setAdditionalFields,

asyncEnqueueStore.setAdditionalFields

)

}

}

}

object AsyncSetAdditionalFields extends TweetStore.AsyncModule {

object Event {

def fromAsyncRequest(

request: AsyncSetAdditionalFieldsRequest,

user: User

): TweetStoreEventOrRetry[Event] =

TweetStoreEventOrRetry(

Event(

additionalFields = request.additionalFields,

userId = request.userId,

optUser = Some(user),

timestamp = Time.fromMilliseconds(request.timestamp)

),

request.retryAction,

RetryEvent

)

}

case class Event(additionalFields: Tweet, userId: UserId, optUser: Option[User], timestamp: Time)

extends AsyncTweetStoreEvent("async\_set\_additional\_fields")

with TweetStoreTweetEvent {

def toAsyncRequest(action: Option[AsyncWriteAction] = None): AsyncSetAdditionalFieldsRequest =

AsyncSetAdditionalFieldsRequest(

additionalFields = additionalFields,

retryAction = action,

userId = userId,

timestamp = timestamp.inMillis

)

override def toTweetEventData: Seq[TweetEventData] =

Seq(

TweetEventData.AdditionalFieldUpdateEvent(

AdditionalFieldUpdateEvent(

updatedFields = additionalFields,

userId = optUser.map(\_.id)

)

)

)

override def enqueueRetry(service: ThriftTweetService, action: AsyncWriteAction): Future[Unit] =

service.asyncSetAdditionalFields(toAsyncRequest(Some(action)))

}

case class RetryEvent(action: AsyncWriteAction, event: Event)

extends TweetStoreRetryEvent[Event] {

override val eventType: AsyncWriteEventType.SetAdditionalFields.type =

AsyncWriteEventType.SetAdditionalFields

override val scribedTweetOnFailure: None.type = None

}

trait Store {

val asyncSetAdditionalFields: FutureEffect[Event]

val retryAsyncSetAdditionalFields: FutureEffect[TweetStoreRetryEvent[Event]]

}

trait StoreWrapper extends Store { self: TweetStoreWrapper[Store] =>

override val asyncSetAdditionalFields: FutureEffect[Event] = wrap(

underlying.asyncSetAdditionalFields)

override val retryAsyncSetAdditionalFields: FutureEffect[TweetStoreRetryEvent[Event]] = wrap(

underlying.retryAsyncSetAdditionalFields)

}

object Store {

def apply(

replicatingStore: ReplicatingTweetStore,

eventBusEnqueueStore: TweetEventBusStore

): Store = {

val stores: Seq[Store] = Seq(replicatingStore, eventBusEnqueueStore)

def build[E <: TweetStoreEvent](extract: Store => FutureEffect[E]): FutureEffect[E] =

FutureEffect.inParallel[E](stores.map(extract): \_\*)

new Store {

override val asyncSetAdditionalFields: FutureEffect[Event] = build(

\_.asyncSetAdditionalFields)

override val retryAsyncSetAdditionalFields: FutureEffect[TweetStoreRetryEvent[Event]] =

build(\_.retryAsyncSetAdditionalFields)

}

}

}

}

object ReplicatedSetAdditionalFields extends TweetStore.ReplicatedModule {

case class Event(additionalFields: Tweet)

extends ReplicatedTweetStoreEvent("replicated\_set\_additional\_fields")

trait Store {

val replicatedSetAdditionalFields: FutureEffect[Event]

}

trait StoreWrapper extends Store { self: TweetStoreWrapper[Store] =>

override val replicatedSetAdditionalFields: FutureEffect[Event] = wrap(

underlying.replicatedSetAdditionalFields)

}

object Store {

def apply(cachingTweetStore: CachingTweetStore): Store = {

new Store {

override val replicatedSetAdditionalFields: FutureEffect[Event] =

cachingTweetStore.replicatedSetAdditionalFields

}

}

}

}