package com.twitter.tweetypie.federated

import com.twitter.ads.internal.pcl.service.CallbackPromotedContentLogger

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.scrooge.ThriftStructFieldInfo

import com.twitter.servo.util.Gate

import com.twitter.strato.catalog.Catalog

import com.twitter.strato.client.Client

import com.twitter.strato.fed.StratoFed

import com.twitter.strato.thrift.ScroogeConv

import com.twitter.tweetypie.ThriftTweetService

import com.twitter.tweetypie.Tweet

import com.twitter.tweetypie.backends.Gizmoduck

import com.twitter.tweetypie.federated.columns.\_

import com.twitter.tweetypie.federated.context.GetRequestContext

import com.twitter.tweetypie.federated.prefetcheddata.PrefetchedDataRepositoryBuilder

import com.twitter.tweetypie.federated.promotedcontent.TweetPromotedContentLogger

import com.twitter.tweetypie.repository.UnmentionInfoRepository

import com.twitter.tweetypie.repository.VibeRepository

import com.twitter.util.Activity

import com.twitter.util.logging.Logger

object StratoCatalogBuilder {

def catalog(

thriftTweetService: ThriftTweetService,

stratoClient: Client,

getUserResultsById: Gizmoduck.GetById,

callbackPromotedContentLogger: CallbackPromotedContentLogger,

statsReceiver: StatsReceiver,

enableCommunityTweetCreatesDecider: Gate[Unit],

): Activity[Catalog[StratoFed.Column]] = {

val log = Logger(getClass)

val getRequestContext = new GetRequestContext()

val prefetchedDataRepository =

PrefetchedDataRepositoryBuilder(getUserResultsById, statsReceiver)

val unmentionInfoRepository = UnmentionInfoRepository(stratoClient)

val vibeRepository = VibeRepository(stratoClient)

val tweetPromotedContentLogger =

TweetPromotedContentLogger(callbackPromotedContentLogger)

// A stitch group builder to be used for Federated Field Column requests. The handler must be the same across

// all Federated Field Columns to ensure requests are batched across columns for different fields

val federatedFieldGroupBuilder: FederatedFieldGroupBuilder.Type = FederatedFieldGroupBuilder(

thriftTweetService.getTweetFields)

val columns: Seq[StratoFed.Column] = Seq(

new UnretweetColumn(

thriftTweetService.unretweet,

getRequestContext,

),

new CreateRetweetColumn(

thriftTweetService.postRetweet,

getRequestContext,

prefetchedDataRepository,

tweetPromotedContentLogger,

statsReceiver

),

new CreateTweetColumn(

thriftTweetService.postTweet,

getRequestContext,

prefetchedDataRepository,

unmentionInfoRepository,

vibeRepository,

tweetPromotedContentLogger,

statsReceiver,

enableCommunityTweetCreatesDecider,

),

new DeleteTweetColumn(

thriftTweetService.deleteTweets,

getRequestContext,

),

new GetTweetFieldsColumn(thriftTweetService.getTweetFields, statsReceiver),

new GetStoredTweetsColumn(thriftTweetService.getStoredTweets),

new GetStoredTweetsByUserColumn(thriftTweetService.getStoredTweetsByUser)

)

// Gather tweet field ids that are eligible to be federated field columns

val federatedFieldInfos =

Tweet.fieldInfos

.filter((info: ThriftStructFieldInfo) =>

FederatedFieldColumn.isFederatedField(info.tfield.id))

// Instantiate the federated field columns

val federatedFieldColumns: Seq[FederatedFieldColumn] =

federatedFieldInfos.map { fieldInfo: ThriftStructFieldInfo =>

val path = FederatedFieldColumn.makeColumnPath(fieldInfo.tfield)

val stratoType = ScroogeConv.typeOfFieldInfo(fieldInfo)

log.info(f"creating federated column: $path")

new FederatedFieldColumn(

federatedFieldGroupBuilder,

thriftTweetService.setAdditionalFields,

stratoType,

fieldInfo.tfield,

)

}

// Instantiate the federated V1 field columns

val federatedV1FieldColumns: Seq[FederatedFieldColumn] =

federatedFieldInfos

.filter(f => FederatedFieldColumn.isMigrationFederatedField(f.tfield))

.map { fieldInfo: ThriftStructFieldInfo =>

val v1Path = FederatedFieldColumn.makeV1ColumnPath(fieldInfo.tfield)

val stratoType = ScroogeConv.typeOfFieldInfo(fieldInfo)

log.info(f"creating V1 federated column: $v1Path")

new FederatedFieldColumn(

federatedFieldGroupBuilder,

thriftTweetService.setAdditionalFields,

stratoType,

fieldInfo.tfield,

Some(v1Path)

)

}

// Combine the dynamic and hard coded federated columns

val allColumns: Seq[StratoFed.Column] =

columns ++ federatedFieldColumns ++ federatedV1FieldColumns

Activity.value(

Catalog(

allColumns.map { column =>

column.path -> column

}: \_\*

))

}

}