package com.twitter.product\_mixer.core.controllers

import com.twitter.context.TwitterContext

import com.twitter.context.thriftscala.Viewer

import com.twitter.product\_mixer.TwitterContextPermit

import com.twitter.product\_mixer.core.model.marshalling.request.ClientContext

/\*\*

\* Mixes in support to forge the UserIds in TwitterContext for debug purposes.

\*

\* A thrift controller can extend DebugTwitterContext and wrap it's execution logic:

\*

\* {{{

\* withDebugTwitterContext(request.clientContext) {

\* Stitch.run(...)

\* }

\* }}}

\*/

trait DebugTwitterContext {

private val ctx = TwitterContext(TwitterContextPermit)

/\*\*

\* Wrap some function in a debug TwitterContext with hardcoded userIds

\* to the ClientContext.userId.

\*

\* @param clientContext - A product mixer request client context

\* @param f The function to wrap

\*/

def withDebugTwitterContext[T](clientContext: ClientContext)(f: => T): T = {

ctx.let(

forgeTwitterContext(

clientContext.userId

.getOrElse(throw new IllegalArgumentException("missing required field: user id")))

)(f)

}

// Generate a fake Twitter Context for debug usage.

// Generally the TwitterContext is created by the API service, and Strato uses it for permission control.

// When we use our debug endpoint, we instead create our own context so that Strato finds something useful.

// We enforce ACLs directly via Thrift Web Forms' permission system.

private def forgeTwitterContext(userId: Long): Viewer = {

Viewer(

auditIp = None,

ipTags = Set.empty,

userId = Some(userId),

guestId = None,

clientApplicationId = None,

userAgent = None,

locationToken = None,

authenticatedUserId = Some(userId),

guestToken = None

)

}

}