package com.sharehub.sharehub.sharehub

import android.annotation.SuppressLint

import android.app.AlertDialog

import android.content.Intent

import android.os.Bundle

import android.text.InputType

import android.view.Gravity

import android.view.View

import android.widget.\*

import com.loopj.android.http.AsyncHttpClient

import com.loopj.android.http.JsonHttpResponseHandler

import com.loopj.android.http.RequestParams

import cz.msebera.android.httpclient.Header

import kotlinx.android.synthetic.main.community.\*

import org.json.JSONObject

import android.widget.AbsListView

import java.nio.file.DirectoryStream

class Community : ShareHubActivityLayoutBase() {

@SuppressLint("SetTextI18n", "RtlHardcoded")

override fun onCreate(savedInstanceState: Bundle?) {

setContentView(R.layout.community)

super.onCreate(savedInstanceState)

val bundle = intent.extras

if (bundle == null) {

replaceActivity(Intent(this, ProfileLauncher::class.java))

return

}

val communityData = bundle.getSerializable("communityData") as CommunityLauncher.CommunityData

community\_header.text = communityData.name

community\_invite\_button.setOnClickListener { view ->

run {

//TODO:

Statics.comingSoon(this)

}

}

for (i in 0 until communityData.users!!.size) {

val view = TextView(this)

view.text = communityData.users!![i]

community\_users.addView(view)

}

for (i in 0 until communityData.boxKey!!.size) {

val entryView = LinearLayout(this)

val layoutParams = LinearLayout.LayoutParams(LinearLayout.LayoutParams.WRAP\_CONTENT, LinearLayout.LayoutParams.WRAP\_CONTENT)

layoutParams.resolveLayoutDirection(LinearLayout.HORIZONTAL)

entryView.layoutParams = layoutParams

val textView = TextView(this)

textView.text = communityData.boxName!![i] + " in " + communityData.boxLocation!![i]

entryView.addView(textView)

val renameButton = Button(this)

renameButton.text = "rename"

renameButton.setOnClickListener { view ->

run {

val builder = AlertDialog.Builder(this)

builder.setTitle("new name")

val input = EditText(this)

input.inputType = InputType.TYPE\_CLASS\_TEXT or InputType.TYPE\_TEXT\_VARIATION\_PASSWORD

builder.setView(input)

builder.setPositiveButton("OK") { dialog, which ->

run {

val context = this@run;

val renameClient = AsyncHttpClient()

val renameParams = RequestParams()

renameParams.add("userKey", Statics.UserKey)

renameParams.add("boxKey", communityData.boxKey!![i])

renameParams.add("newName", input.text.toString())

renameClient.post(Constants.BoxGateway.Rename, renameParams, object : JsonHttpResponseHandler() {

override fun onSuccess(statusCode: Int, headers: Array<out Header>?, response: JSONObject) {

try {

if (response["success"] as Boolean) {

textView.text = input.text.toString() + " in " + communityData.boxLocation!![i]

}

else {

Toast.makeText(context, "you have been logged out", Toast.LENGTH\_LONG).show()

Statics.logout(context)

context.replaceActivity(Intent(context, Login::class.java))

}

}

catch (e: Exception) {

Toast.makeText(context, "an exception occurred: " + e.message, Toast.LENGTH\_LONG).show()

}

}

override fun onFailure(statusCode: Int, headers: Array<out Header>?, responseString: String?, throwable: Throwable?) {

Toast.makeText(context, "4\_http request error: " + throwable?.message, Toast.LENGTH\_LONG).show()

}

})

}

}

builder.setNegativeButton("Cancel") { dialog, which -> dialog.cancel() }

builder.show()

}

}

entryView.addView(renameButton)

val giveButton = Button(this)

giveButton.text = "give"

giveButton.setOnClickListener { view ->

run {

//TODO:

Statics.comingSoon(this)

}

}

entryView.addView(giveButton)

community\_boxes.addView(entryView)

}

community\_chat\_container.visibility = View.GONE

var messages: Array<Message> = arrayOf()

fun filter() {

val unfiltered = messages

val filtered = mutableListOf<Message>()

for (i in 0 until unfiltered.size) {

var has: Boolean = false

for (j in 0 until filtered.size) {

if (unfiltered[i].objectId!!.equals(filtered[j])) {

has = true

}

}

if (!has) {

filtered.add(unfiltered[i])

}

}

messages = filtered.toTypedArray()

}

fun loadMessages(skip: Int, limit: Int, callback: (messages: Array<Message>) -> Unit) {

val client = AsyncHttpClient()

val params = RequestParams()

params.add("userKey", Statics.UserKey)

params.add("communityId", communityData.uniqueName)

params.add("skip", skip.toString())

params.add("limit", limit.toString())

client.post(Constants.CommunityGateway.GetMessage, params, object : JsonHttpResponseHandler() {

override fun onSuccess(statusCode: Int, headers: Array<out Header>?, response: JSONObject) {

if (response.getBoolean("success")) {

val localMessages = Message.parseArray(response, "messages")

if (localMessages == null) {

Toast.makeText(this@Community, "json parsing error occurred", Toast.LENGTH\_LONG).show()

}

else {

messages = arrayOf(\*messages, \*localMessages)

filter()

callback(messages)

}

}

else {

Toast.makeText(this@Community, "an error occurred: " + response.getString("message"), Toast.LENGTH\_LONG).show()

}

}

override fun onFailure(statusCode: Int, headers: Array<out Header>?, responseString: String?, throwable: Throwable?) {

Toast.makeText(this@Community, "an http error occurred: " + throwable!!.message, Toast.LENGTH\_LONG).show()

}

})

}

fun loadUI(response: Array<Message>) {

community\_chat\_container.visibility = View.VISIBLE

community\_chat.removeAllViews()

for (i in response.size - 1 downTo 0) {

val entryView = LinearLayout(this)

val layoutParams = LinearLayout.LayoutParams(LinearLayout.LayoutParams.WRAP\_CONTENT, LinearLayout.LayoutParams.WRAP\_CONTENT)

layoutParams.resolveLayoutDirection(LinearLayout.HORIZONTAL)

layoutParams.gravity = if (response[i].author.equals(Statics.Username)) {

Gravity.RIGHT

}

else {

Gravity.LEFT

}

entryView.layoutParams = layoutParams

val textView = TextView(this)

textView.text = response[i].content

entryView.addView(textView)

community\_chat.addView(entryView)

}

community\_chat\_scroll.fullScroll(View.FOCUS\_DOWN)

}

community\_chat\_scroll.viewTreeObserver.addOnScrollChangedListener {

if (!canScrollUp(community\_chat\_scroll)) {

loadMessages(messages.size, messages.size + 10, { response -> loadUI(response) })

}

}

fun initRestMessage() {

val client = AsyncHttpClient()

val params = RequestParams()

params.add("userKey", Statics.UserKey)

params.add("communityId", communityData.uniqueName)

client.post(Constants.CommunityGateway.RESTMessage, params, object : JsonHttpResponseHandler() {

override fun onSuccess(statusCode: Int, headers: Array<out Header>?, response: JSONObject) {

if (response.getBoolean("success")) {

val message = Message.parse(response, "restMessage")!!

val list = messages.toMutableList()

list.reverse()

list.add(message)

list.reverse()

messages = list.toTypedArray()

filter()

loadUI(messages)

}

else {

}

initRestMessage()

}

override fun onFailure(statusCode: Int, headers: Array<out Header>?, responseString: String?, throwable: Throwable?) {

Toast.makeText(this@Community, "an http error occurred: " + throwable!!.message, Toast.LENGTH\_LONG).show()

initRestMessage()

}

})

}

community\_chat\_send\_button.setOnClickListener { view ->

run {

val client = AsyncHttpClient()

val params = RequestParams()

params.add("userKey", Statics.UserKey)

params.add("communityId", communityData.uniqueName)

params.add("message", community\_chat\_textbox.text.toString())

client.post(Constants.CommunityGateway.SendMessage, params, object: JsonHttpResponseHandler(){

override fun onSuccess(statusCode: Int, headers: Array<out Header>?, response: JSONObject) {

if(!response.getBoolean("success")){

Toast.makeText(this@Community, "sending message failed: " + response.getString("message"), Toast.LENGTH\_LONG).show()

}

}

override fun onFailure(statusCode: Int, headers: Array<out Header>?, responseString: String?, throwable: Throwable?) {

Toast.makeText(this@Community, "an http error occurred: " + throwable!!.message, Toast.LENGTH\_LONG).show()

}

})

}

}

initRestMessage()

}

private fun canScrollUp(view: View): Boolean {

return if (view is AbsListView) {

view.childCount > 0 && (view.firstVisiblePosition > 0 || view.getChildAt(0).top < view.paddingTop)

}

else {

view.scrollY > 0

}

}

}