package com.noname.whatsapp.tracker

import java.sql.Types

import java.util.concurrent.ConcurrentHashMap

////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////

// создать список активностей

1. //++++ POST: /api/createActivities

private Collection<ActivityRecord> records

public class ActivityRecord {

private Integer subscriptionId;

private LocalDateTime actualDate;

private Boolean isOnline;

}

Response:

public class CreatedActivity {

private Integer code;

private String message;

}

//------------------------------------------------------------------------------

fun insertPresenceNoteBulkCall(listPhoneDBIdOnIsOnline: ArrayList<Pair<Int, Boolean>>) {

try {

OracleHikariDataSource.dataSource.connection.use {

val sqlQuery =

"insert into visitnote (phone\_id, is\_online, add\_time) values (?, ?, cast(sys\_extract\_utc(systimestamp) as date)) LOG ERRORS REJECT LIMIT UNLIMITED"

it.prepareStatement(sqlQuery).use { ps ->

for (pair in listPhoneDBIdOnIsOnline) {

ps.setInt(1, pair.first)

ps.setBoolean(2, pair.second)

ps.addBatch()

}

ps.executeBatch()

}

}

} catch (ex: Exception) {

DBManager.log.error("error bulk inserting visitnotes in DB ${ex.message}")

}

}

////////////////////////////////////////////////////////////////////////////////

// получить список агентов (ботов) в конкретном статусе (agentStatusId)

2. //++++ GET: /api/getAgentsList

QueryParams:

//----------------------------------------------------

final String agentStatus = QueryParam("agentStatusId");

0,reserved

1,supported

3,tracking

4,banned

5,quarantine

//------------------------------------------------------------------------------

fun getAuthkeysByGroup(groupName: String) =

OracleHikariDataSource.dataSource.connection.use{

var sqlQuery = "s**elect authkey.phone\_num, authkey.payload from authkey join botgroup on (authkey.group\_id=botgroup.id) where botgroup.name = ?**"

it.prepareStatement(sqlQuery).use{ps ->

ps.setString(1, groupName)

ps.executeQuery().use {rs ->

generateSequence {

if (rs.next()) Pair(rs.getLong(1), rs.getString(2)) else null

}

.filterNotNull()

.map {

Authkey(it.first, it.second)

}

.toList()

}

}

}

////////////////////////////////////////////////////////////////////////////////

// заменить забаненного агента\бота на конкретном номере телефона (м.б. несколько подпискок) и вернуть PayLoad нового агента\бота

3. //++++ POST: /api/updateAgentStatus

QueryParams:

//----------------------------------------------------

final String phoneNum = QueryParam("phoneNum");

final String agentStatus = QueryParam("agentStatusId");

0,reserved

1,supported

3,tracking

4,banned

5,quarantine

Response:

return\_val:=replacement\_phone||':'||replacement\_payload;

//------------------------------------------------------------------------------

fun replaceBannedAuthkey(phoneNum: Long, botType: BotType): String?=

try {

OracleHikariDataSource.dataSource.connection.use {

val isTracker = botType == BotType.TRACKER

it.prepareCall("begin ? := wa\_tracking.ban\_bot(?,?); end;").use { cstmt ->

cstmt.registerOutParameter(1, Types.VARCHAR)

cstmt.setLong(2, phoneNum)

cstmt.setBoolean(3, isTracker)

cstmt.execute()

cstmt.getString(1)

}

}

}

catch(ex: Exception){

DBManager.log.error("error replacing banned authkey", ex)

null

}

////////////////////////////////////////////////////////////////////////////////

// заменить карантинного агента\бота на конкретном номере телефона (м.б. несколько подпискок) и вернуть PayLoad нового агента\бота

4. //++++ POST: /api/updateAgentStatus

QueryParams:

//----------------------------------------------------

final String phoneNum = QueryParam("phoneNum");

final String agentStatus = QueryParam("agentStatusId");

0,reserved

1,supported

3,tracking

4,banned

5,quarantine

Response:

return\_val:=replacement\_phone||':'||replacement\_payload;

//------------------------------------------------------------------------------

fun replaceQuarantinedAuthkey(phoneNum: Long, botType: BotType): String?=

try {

OracleHikariDataSource.dataSource.connection.use {

val isTracker = botType == BotType.TRACKER

it.prepareCall("begin ? := wa\_tracking.quarantine\_bot(?,?); end;").use { cstmt ->

cstmt.registerOutParameter(1, Types.VARCHAR)

cstmt.setLong(2, phoneNum)

cstmt.setBoolean(3, isTracker)

cstmt.execute()

cstmt.getString(1)

}

}

}

catch(ex: Exception){

DBManager.log.error("error replacing quarantined authkey", ex)

null

}

////////////////////////////////////////////////////////////////////////////////

// Получить список подписок\телефонов, которые обслуживаются конкретный агентов/ботом (с признаками isFresh, botIsFresh). Нахождение агента\бота по его телефонному номеру в WattsApp

5. //++++ GET: /api/getSubsriptionsListByAgent

QueryParams:

//------------------------------------------------------------------------------

final String agentPhoneNum = QueryParam("agentPhoneNum");

final Integer considerBotFreshHours = QueryParam("considerBotFreshHours"); - **??? considerBotFreshHours**

private Collection<SubscriptionRecord> records

public class SubscriptionRecord {

private String agentPhoneNum;

private String subscriptionPhoneNum;

private Boolean subscriptionPhoneIsFresh;

private Boolean agentPhoneisFresh;

}

//------------------------------------------------------------------------------

data class Quadruple<out A,out B, out C, out D>(val first:A, val second:B, val third:C, val fourth:D)

fun getSubscriptionNumbersForBot(phoneNum: Long, considerBotFreshHours: Int) :AuthkeyConfig?=

try {

OracleHikariDataSource.dataSource.connection.use {

var sqlQuery =

"""**select authkey.phone\_num,**

**| subscriptionphone.phone\_num,**

**| subscriptionphone.id,**

**| case when cast(sys\_extract\_utc(systimestamp) as date)-subscriptionphone.add\_time<?/24 then 1 else 0 end as isFresh,**

**| case when cast(sys\_extract\_utc(systimestamp) as date)-authkey.last\_change\_time<?/24 then 1 else 0 end as botIsFresh**

**| from authkey join subscriptionphone on (subscriptionphone.key\_id=authkey.id) where authkey.phone\_num = ?**""".trimMargin()

it.prepareStatement(sqlQuery).use { ps ->

ps.setInt(1,considerBotFreshHours)

ps.setInt(2,considerBotFreshHours)

ps.setLong(3, phoneNum)

ps.executeQuery().use { rs ->

generateSequence {

if (rs.next()) Quadruple(rs.getLong(1), rs.getLong(2), rs.getInt(3), rs.getBoolean(4) or rs.getBoolean(5)) else null

}

.filterNotNull()

.groupBy { it.first }

.mapValues {

ConcurrentHashMap<Long, Pair<Int,Boolean>>().apply {

for (listItem in it.value)

put(listItem.second, Pair(listItem.third,listItem.fourth))

}

}

.map {

AuthkeyConfig(it.key, it.value)

}

.firstOrNull()

}

}

}

}

catch(ex: Exception){

DBManager.log.error("error getting subscription numbers for bot", ex)

null

}

////////////////////////////////////////////////////////////////////////////////

// изменить статус подписки

6. //++++ POST: /api/updateSubscriptionStatus

QueryParams:

//----------------------------------------------------

final String subscriptionId = QueryParam("subscriptionId");

final String subscriptionStatus = QueryParam("subscriptionStatusId");

-2, Phone not exists

-1, Cancelled

0, Actual

1, Closed

Response:

public class updatedStatus {

private Integer code;

private String message;

}

//------------------------------------------------------------------------------

fun updateSubscriptionNumberState(subscriptionPhoneId: Int, isValid: Boolean) =

try {

OracleHikariDataSource.dataSource.connection.use {

it.prepareCall("call wa\_tracking.update\_subscription\_state(?,?)").use { cstmt ->

cstmt.setInt(1, subscriptionPhoneId)

cstmt.setBoolean(2, isValid)

cstmt.executeUpdate()

}

}

}

catch(ex: Exception){

DBManager.log.error("error updating subscriptions state", ex)

}

////////////////////////////////////////////////////////////////////////////////

// изменить аватар на подписке

7. //++++ POST: /api/updateSubscriptionAvatar

QueryParams:

//----------------------------------------------------

final String subscriptionId = QueryParam("subscriptionId");

public class AvatarRecord {

private byte[] avatar;

}

Response:

public class updatedAvatar {

private Integer code;

private String message;

}

//------------------------------------------------------------------------------

fun updateAvatar(phoneNum: Long, avatarID: Long, rawImage: ByteArray) {

try {

OracleHikariDataSource.dataSource.connection.use {

it.prepareStatement("update subscriptionphone set avatar\_id = ?, avatar = ? where phone\_num = ?")

.use { stmt ->

stmt.setLong(1, avatarID)

stmt.setBytes(2, rawImage)

stmt.setLong(3, phoneNum)

stmt.executeUpdate()

}

}

}

catch(ex: Exception){

DBManager.log.error("error updating avatar", ex)

}

}

////////////////////////////////////////////////////////////////////////////////

// получить список изменений по всем подпискам начиная с даты {dd.mm.yyyy}

8. // +++++ GET: /api/getModifiedSubscriptions

QueryParams:

//----------------------------------------------------

final LocalDateTime actualDate = QueryParam("actualDate")));

Response:

private Collection<UserSubscriptionInfo> records;

public class UserSubscriptionInfo {

private LocalDateTime actualDate;

private String subscriptionName;

private String phoneNum;

private Byte subscriptionStatusId;

private Boolean onlineNotify;

}

//------------------------------------------------------------------------------

// 4 delete

//------------------------------------------------------------------------------

fun getNewSubscriptionData(rowID: String) =

try {

OracleHikariDataSource.dataSource.connection.use {

it.prepareStatement("**select subscriptionphone.phone\_num, subscriptionphone.id, subscriptionphone.is\_valid from subscriptionphone where subscriptionphone.rowid = ?**")

.use { stmt ->

stmt.setString(1, rowID)

stmt.executeQuery().use {

if (it.next())

Triple(it.getLong(1), it.getInt(2), it.getInt(3))

else

null

}

}

}

}

catch(ex: Exception){

DBManager.log.error("error getting new subscription data", ex)

null

}

////////////////////////////////////////////////////////////////////////////////

// получить список всех подписок, по которым требуется уведомления пушами

9. // +++++ GET: /api/getNotificationsList

Response:

private Collection<NotificationRecord> records;

public class NotificationRecord {

private String gcmToken;

private String phoneNum;

private String subscriptionName;

private String appName;

}

//------------------------------------------------------------------------------

**// 17817 записей на реале -- как часто будет вызываться?**

//------------------------------------------------------------------------------

fun getNotificationMap() =

OracleHikariDataSource.dataSource.connection.use {

it.prepareStatement("**select appuser.gcm\_token, subscriptionphone.phone\_num, subscriptionphone.assigned\_name, appuser.app\_name from appuser join subscriptionphone on (appuser.id=subscriptionphone.user\_id) where notify=1 and key\_id is not null and is\_removed=0"**)

.use { stmt ->

val resultMap = HashMap<Long, MutableSet<Triple<String, String, String>>>()

stmt.executeQuery().use {

while (it.next()) {

resultMap.getOrPut(it.getLong(2)) { mutableSetOf() }.add(

Triple(

it.getString(1),

it.getString(3),

it.getString(4) ?: "com.peanutbutter.wastat"

)

)

}

resultMap

}

}

}

////////////////////////////////////////////////////////////////////////////////

10. // ВЕРНУТЬ СТАТУС КОНКРЕТНОЙ ПОДПИСКИ НЕЗАВИСИМО ОТ СТАТУСА

//------------------------------------------------------------------------------

// (!!!) запрос вернет 0 записей, если подписка не имеет обслуживающего агента\бота [key\_id is null]

//------------------------------------------------------------------------------

fun getNotifDeliveryStatus(rowSubscriptionPhoneID: String) =

try {

OracleHikariDataSource.dataSource.connection.use {

it.prepareStatement(**"select notify, phone\_num, gcm\_token, assigned\_name, app\_name from subscriptionphone join appuser on (appuser.id=subscriptionphone.user\_id) where subscriptionphone.rowid = ? and key\_id is not null"**)

.use { stmt ->

stmt.setString(1, rowSubscriptionPhoneID)

stmt.executeQuery().use {

if (it.next())

Triple(

it.getInt(1),

it.getLong(2),

Triple(it.getString(3), it.getString(4), it.getString(5) ?: "com.peanutbutter.wastat")

)

else null

}

}

}

}

catch(ex: Exception){

DBManager.log.error("error getting notification delivery status", ex)

null

}

////////////////////////////////////////////////////////////////////////////////

11. // ВЕРНУТЬ СПИСОК АКТИВНЫХ ПОДПИСОК ПОЛЬЗОВАТЕЛЯ (removed=0, where appuser.rowid = ?)

//------------------------------------------------------------------------------

// (!!!) запрос вернет 0 записей, если подписка не имеет обслуживающего агента\бота[key\_id is null]

//------------------------------------------------------------------------------

fun getNotifDeliveryStatusForUser(rowAppUserID: String) =

try {

OracleHikariDataSource.dataSource.connection.use {

it.prepareStatement(**"select phone\_num, gcm\_token, assigned\_name, app\_name from subscriptionphone join appuser on (appuser.id=subscriptionphone.user\_id) where appuser.rowid = ? and key\_id is not null and is\_removed=0"**)

.use { stmt ->

val result = mutableListOf<Pair<Long, Triple<String, String, String>>>()

stmt.setString(1, rowAppUserID)

stmt.executeQuery().use {

while (it.next())

result.add(

Pair(

it.getLong(1),

Triple(

it.getString(2),

it.getString(3),

it.getString(4) ?: "com.peanutbutter.wastat"

)

)

)

}

result

}

}

}

catch(ex: Exception){

DBManager.log.error("error getting notification delivery status for user", ex)

mutableListOf<Pair<Long, Triple<String, String, String>>>()

}

////////////////////////////////////////////////////////////////////////////////

12. // ВЕРНУТЬ СОСТОЯНИЕ КОНКРЕТНОЙ ПОДПИСКИ (where subscriptionphone.rowid = ?)

fun getUpdatedSubscriptionActiveState(rowID: String) =

try {

OracleHikariDataSource.dataSource.connection.use {conn ->

var result =

conn.prepareStatement(**"select authkey.phone\_num, subscriptionphone.phone\_num, subscriptionphone.id, subscriptionphone.is\_removed from subscriptionphone join authkey on (subscriptionphone.key\_id=authkey.id) where subscriptionphone.rowid = ?"**)

.use { stmt ->

stmt.setString(1, rowID)

stmt.executeQuery().use {

if (it.next()) {

var quad = Quadruple(it.getLong(1), it.getLong(2), it.getInt(3), it.getBoolean(4))

if (quad.fourth) {

var sql = "update subscriptionphone set key\_id=null where id=?"

conn.prepareStatement(sql).use { ps ->

ps.setInt(1, quad.third)

ps.executeUpdate()

}

}

quad

}

else null

}

}

result

}

}

catch(ex: Exception){

DBManager.log.error("error getting updated subscription active state", ex)

null

}

////////////////////////////////////////////////////////////////////////////////

13. // ВЕРНУТЬ НЕОБРАБОТАННЫЕ ПОДПИСКИ В ТЕЧЕНИЕ ХХ часов (is\_valid = null)

fun getUnprocessedPhonesCall() =

try {

OracleHikariDataSource.dataSource.connection.use {

it.prepareStatement(**"select phone\_num, id from (select phone\_num, id, add\_time, cast(sys\_extract\_utc(systimestamp) as date) as cur\_time from subscriptionphone where is\_valid is null) where (cur\_time-add\_time)\*24\*60\*60>120"**)

.use { stmt ->

//stmt.fetchSize = 1000

val resultList = ArrayList<Triple<Long, Int, Int>>()

stmt.executeQuery().use {

while (it.next()) {

resultList.add(Triple<Long, Int, Int>(it.getLong(1), it.getInt(2), 0))

}

resultList

}

}

}

}

catch(ex: Exception){

DBManager.log.error("error getting unprocessed phones", ex)

ArrayList<Triple<Long, Int, Int>>()

}