AuditLog

Contents

Purpose

Document all changes to tables and keep an auditlog with user who did those, timestamp, current and previous value.

Discussions about this: https://forums.appgini.com/phpbb/viewtopic.php?f=4&t=1369

Disclaimer

We have tried to make the installation as easy as possible. Landinialejandro provided a plugin that makes installation even easier – see below. However, with the certain knowledge that no matter how idiot proof you make something, nature will provide a better idiot, we feel we must add the following:

Despite the fact that this extension was used in several different applications, built with AppGini, we in NO way take ANY responsibility for mistakes in the code or that YOU might make. ALWAYS backup you files and your database before attempting a major modification!

Installation as Plugin-Version (recommended)

This works only if you have acquired some other official AppGini plugin. If you do not have an official plugin or you want to use the manual installation instead, please read below: Manual Installation (not recommended!).

New from version 1.71: The AppGini forum member landinialejandro created a plugin which makes installation much easier. Please see the forum (https://forums.appgini.com/phpbb/viewtopic.php?f=4&t=1369) and/or the github page where he placed it (https://github.com/myappgini/sbm_audit_log)

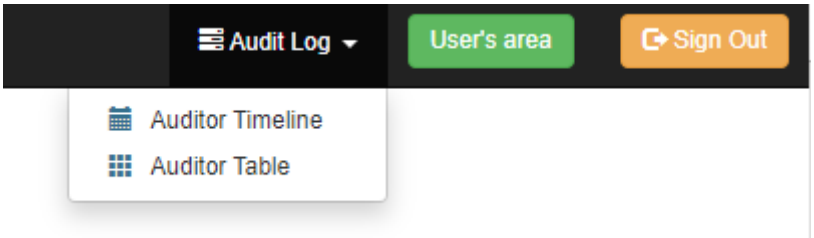
If you decide to use this plugin for installation (**yes, it's the recommended way**),

1. read this documentation
2. download the latest plugin from it's homepage https://github.com/myappgini/sbm_audit_log
3. read the README.md of the plugin.
4. If needed/wished, update the folder app-resources of the plugin by putting all files from this ZIP directly into that folder
5. upload the plugin to your AppGini generated site as usual
6. go to the admin area of your site and install the plugin as usual.

Remember: When you add more tables to your AppGini application after you have installed AuditLog, you need to run the plugin again (of do the manual installation for the new tables)!

Access to Audit Log

Once installed (plugin, or manual with auditor-table-link in the admin area) you as superadmin have the opportunity to see all changes in the application. You will find a new menu in the admin area which should look like this:



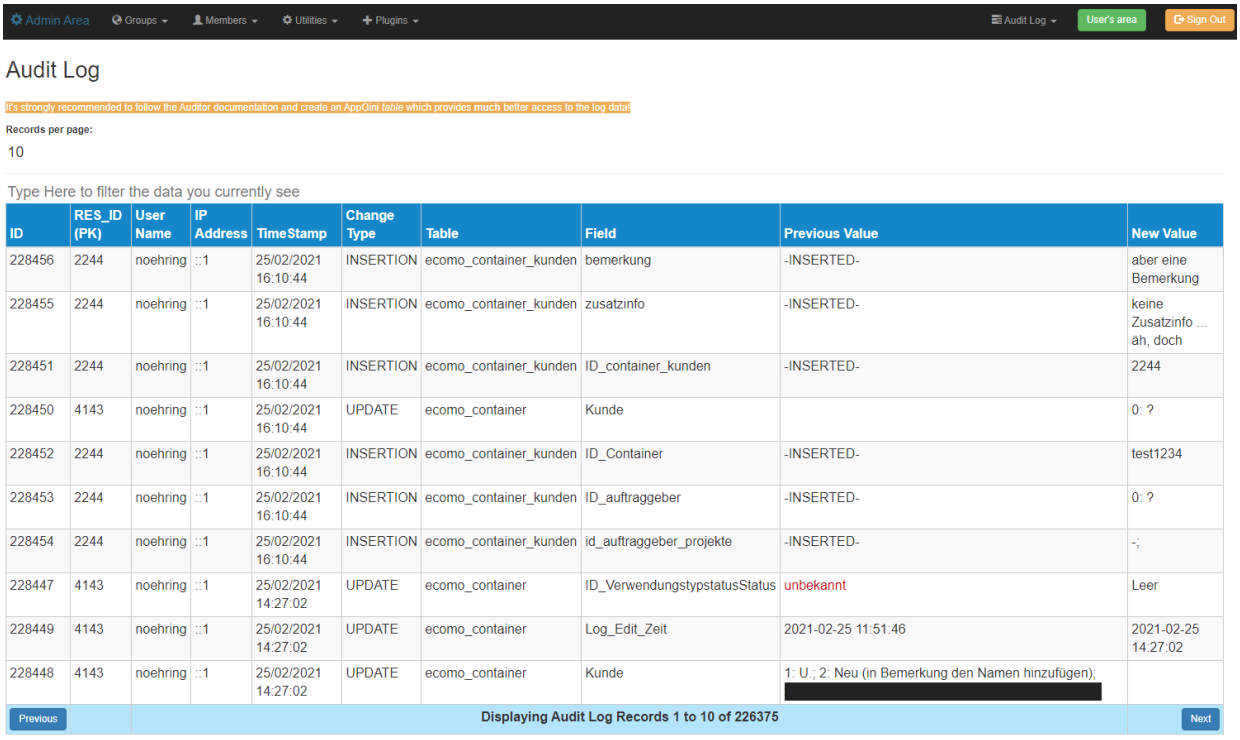
Audit Log menu in the Admin Area

Standard Audit Log Table (installed automatically)

The *Auditor Table* view provides all information, but in a *very basic* form. If you want to have easier access and better filters, please read below: Advanced Audit Log Table (recommended):

You can set the number or records you want to see at once. Just press the enter key after you set the desired number and the page will reload.

Note: The simple search function searches only through the records that are displayed on the page. It does not search the complete auditor log table.



Standard Audit Log in the admin area. Please note, that the IP address is correct. The screenshot was done in a local testing environment, this it's the IPv6 format of the localhost address (IPv4: 127.0.0.1).

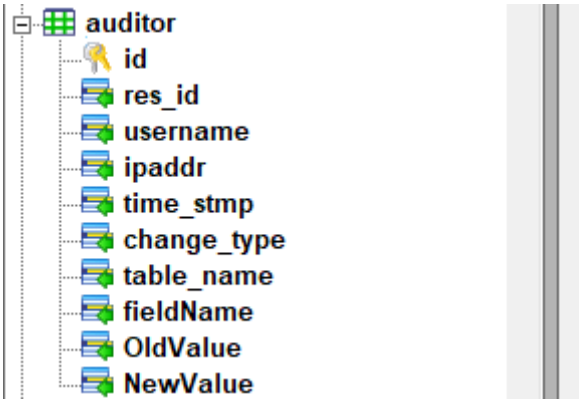
Advanced Audit Log Table (recommended)

You can have a much better display and search capabilities when you add an extra table to your AppGini application.

It's easy! Just create a simple table in your AppGini project and make sure it has the same name and the same fields as the auditor table in the database! You can copy the contents from the file `Clipboard_AppGini_AuditLog_Table.txt` into your clipboard and paste it back into your AppGini application with the AppGini buttons in the toolbar, a new table with the name `auditor` will be created with all fields:

AppGini Paste Table/Field button

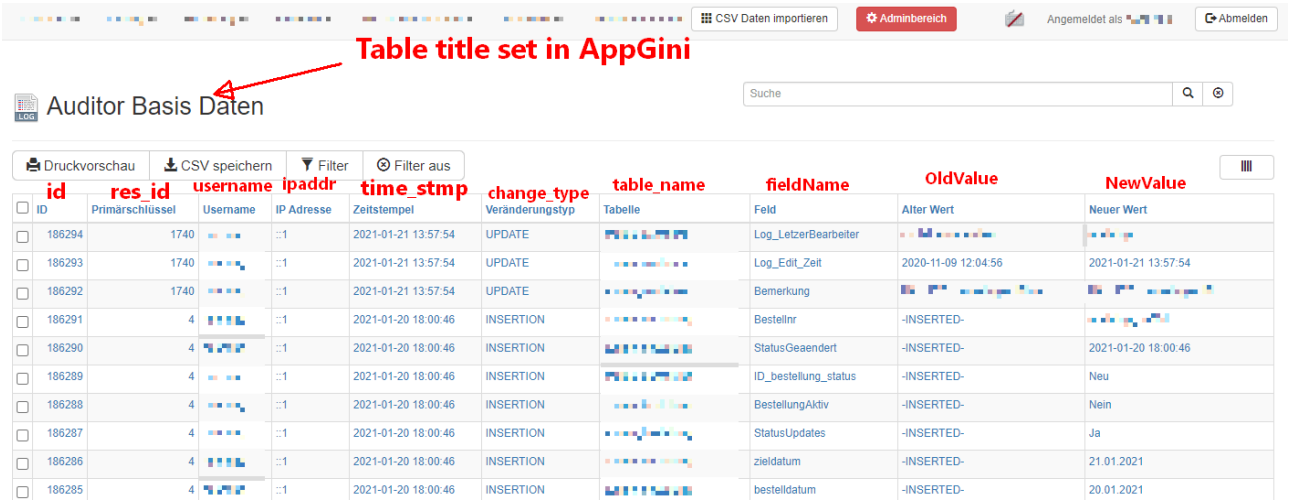
The table will look like this:



Auditor table in your AppGini project.

Of course, you can label the fields as you like (in your own language). When following this, AppGini will create a standard page for the table which you can search and filter with the standard AppGini functions which are far more advanced than the solution above.

Your table could look like this:



Auditor-Table created in the AppGini project and here in the generated application. Red descriptions added to show which labels are the columns from the auditor table.

Set permissions to that Auditor table: **View permission is enough!** It's suggested, that you **do not allow** anyone to insert, edit, update or delete in that table.

Auditor Timeline

The *Auditor Timeline* was added in version 1.9 to this AppGini extension. It allows you to get the state of each record in each table (if all tables use the auditor script) at every change. This means, you can easily see and compare before and after of any record at a given timestamp.

The *Auditor Timeline* provides a wizard which leads you through the process. The steps are as follows:

Step 1: Choose a table from all tables that can be found in the Audit Log.

Step 2: Choose a record (from column res_id). The column res_id shows the primary key from the table you have chosen in step 1. The number that follows the primary key is the amount of timestamps for which changes have been recorded in the auditor for that specific record. Example: 4143 (100) would mean, the record with primary key 4143 has been logged at 100 different times (This is not equal the number of changes, as in one edit more than one field can be changed, resulting in two entries in the auditor, but both carrying the same timestamp.)

From this step on, the field to select the table is locked.

Step 3: Verify you have chosen the correct record. For this the current record is taken from the chosen table and displayed. Lookup fields are showing the values that are actually saved in the database, not those, you would see in the generated AppGini application.

If you have chosen the wrong record, simply press the *Start Over* button at any time.

From this step on, the field to select the record (primary key) is locked.

Step 4: Choose the timeframe for which the timeline should be generated. The wizard will suggest the first timestamp as start and the last as end for the timeline. Keep the date/time-format or you will break the timeline (no checks for dates done in the background).

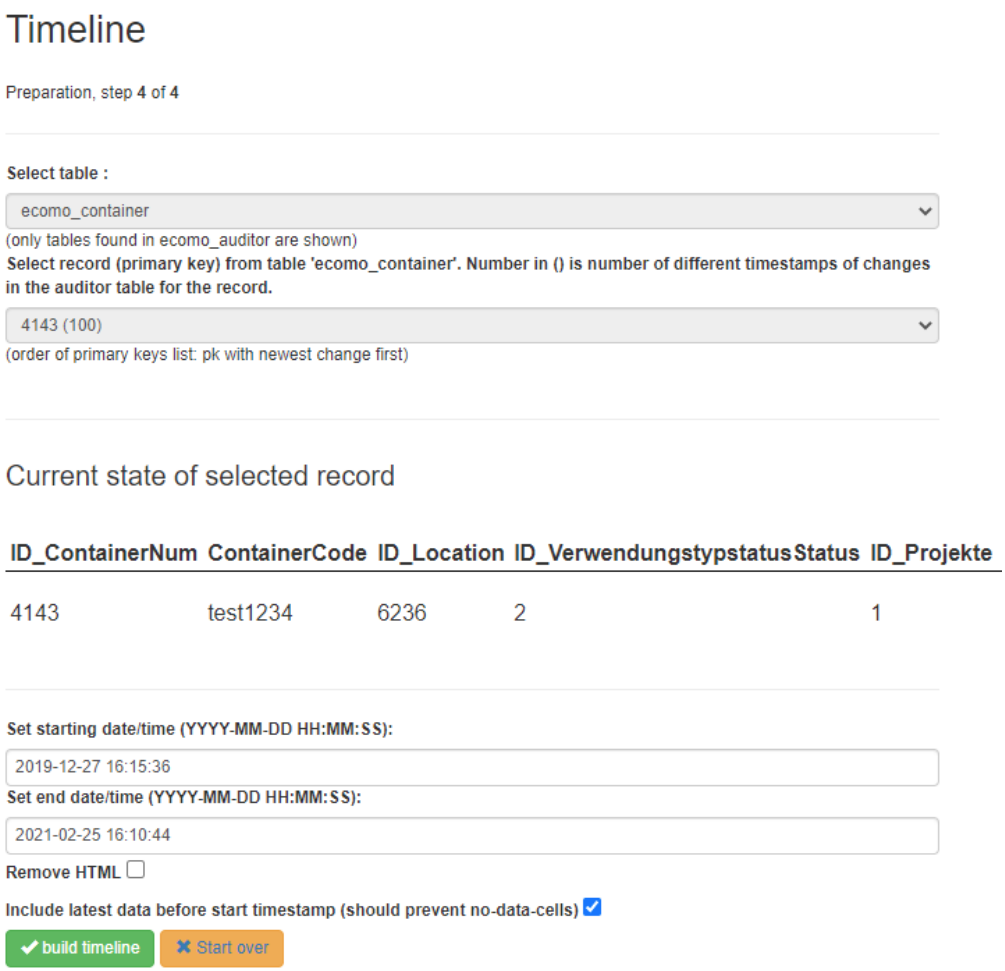
This last step shows some additional options.

Remove HTML: This will use the PHP function strip_tags before the data is displayed and remove all HTML. This might be helpful if the data contains some formatting HTML for example. Default is unchecked.

Include latest data before start timestamp (should prevent no-data-cells): If you check this (default is checked), the timeline will try to pull data for each of the columns from older auditor

entries before the start that is set. This should give you a better idea what the record looked like at a certain point in time.

The next picture shows the last step of the wizard, before the timeline will be generated.



Timeline, Step 4

Once the last step is done and you decided for your options, press the *build timeline*-button.

The result could look like this:

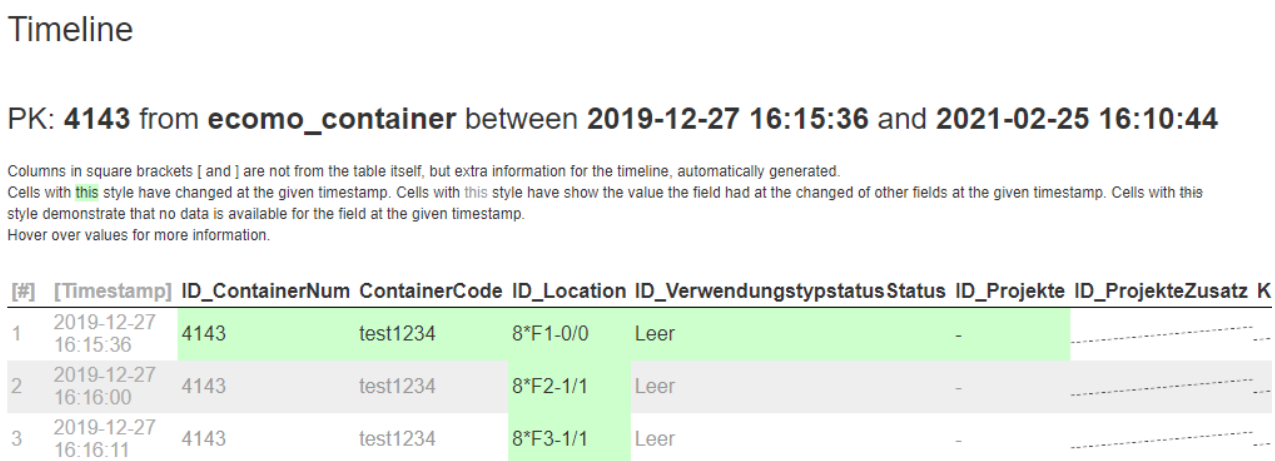
The columns with the [ ] around their name are automatically generated (there are some more on the far right side).

The green fields have been changed at the time that can be seen on the left (column Timestamp). If something was not changed, the previous value will be repeated (gray text) to make it easier to

grasp the state of the record (#2 for example ID_ContainerNum was not changed, but ID_Location was).

You can also see the last column in the picture (ID_ProjekteZusatz). The dashed line shows, that there is no data for that field at that timestamp – not even old data that should have been pulled. One reason could be, the field did not exist at the time of the change. Another reason might be, that changes in this table have not been documented from the very beginning.

You can hover all cells and get more information.



Timeline

PK: **4143** from **ecomo_container** between **2019-12-27 16:15:36** and **2021-02-25 16:10:44**

Columns in square brackets [and] are not from the table itself, but extra information for the timeline, automatically generated.
Cells with this style have changed at the given timestamp. Cells with this style have show the value the field had at the changed of other fields at the given timestamp. Cells with this style demonstrate that no data is available for the field at the given timestamp.
Hover over values for more information.

[#]	[Timestamp]	ID_ContainerNum	ContainerCode	ID_Location	ID_VerwendungstypstatusStatus	ID_Projekte	ID_ProjekteZusatz	K
1	2019-12-27 16:15:36	4143	test1234	8*F1-0/0	Leer	-		
2	2019-12-27 16:16:00	4143	test1234	8*F2-1/1	Leer	-		
3	2019-12-27 16:16:11	4143	test1234	8*F3-1/1	Leer	-		

Timeline

Make the *Auditor Timeline* accessible to your users

If you want to give certain users the permission to use the *Auditor Timeline*, create a new pseudo table in AppGini. Choose your own name for that table (if you copy the code below, name it *auditortimeline*. Make sure you do not copy the footer from this PDF). Once you generated your application, edit the generated file in the root (!) folder of your application.

Note: This file will be overwritten when you regenerate your app. We suggest saving the file to a different folder, so that you can simply copy it back later.

Note: You will need to give access permissions to the group for table *auditortimeline*! **Warning**: Every person (usergroup) having access to this timeline can view all records in all tables!

Replace *all* code in the file and copy and paste the following code into the file:

```
<?php
  session_start();
  $currDir = dirname(__FILE__);
  include("$currDir/defaultLang.php");
  include("$currDir/language.php");
  include("$currDir/lib.php");
```

```
  include_once("$currDir/header.php");

  /* grant access to all users who have access to the orders table */
  $user_can_access = get_sql_from('auditortimeline');
  if(!$user_can_access) exit(error_message('Access denied!', false));

  include("$currDir/hooks/audit/auditLog_timeline.php");

  if(!$footerCode){
     include_once("$currDir/footer.php");
  } else {
     ob_start(); include_once("$currDir/footer.php"); $dFooter=ob_get_contents(); ob
_end_clean();
     echo str_replace('<%%FOOTER%%>', $dFooter, $footerCode);
  }
?>
```

Possible Adjustments

Custom table name for the auditor table (before installation)

If you prefer, you can easily change the name of the auditor table for example to match some table-prefix you are using. This should be done *before* running the installation via plugin as some file contents needs to be changed.
Note: table names are case-sensitive (at least in Linux). Therefor I suggest lowercase table names.

Open `audit_tableSQL.sql`

Search for

```
`auditor` (
```

replace with

```
`your_tablename_here` (
```

Open `/hooks/auditLog_functions.php`

Search for

```
define("AUDITTABLENAME", 'auditor');
```

replace with

```
define("AUDITTABLENAME", 'your_tablename_here');
```

Open `/hooks/auditLog_timeline.php`

Search for

```
define("AUDITTABLENAME", 'auditor');
```

replace with

```
define("AUDITTABLENAME", 'your_tablename_here');
```

Open `/admin/auditLog.php`

Search for

```
define("AUDITTABLENAME", 'auditor');
```

replace with

```
define("AUDITTABLENAME", 'your_tablename_here');
```

Now run the installation using the plugin.

Exact position of audit call functions

The plugin-installation-script (and the Manual Installation (not recommended!) instructions below) place the call for the audit log functions (`table_before_change` and `table_after_change`) directly after the opening of certain functions in the hooks/tablename.php file. You might have the need to adjust the position of the calls of the auditor functions.

For example, you may need to change some data directly after the user clicked the save-changes-button, this in the before_update function – or – that you need to change data after it has been saved. In these cases you need to move the call for the auditor function by hand to a position after your changes.

If you do this, everything should work. Once you run the installation plugin again, the plugin should notice that the call already exist (does not matter where in the function), notify you about the existence during installation, skip the function but continue with the next. If the plugin has a problem it will notify you that you have to do some manual labor.

Extras

Documenting changes for another table: fct Audit_Manually

In version 1.7 of this script a new function was introduced: `Audit_Manually`

The function allows to manual add to Auditor-table (and/or save old value to session variable).

This is useful, in case you have table TA and table TB. TB is a child of TA (holds the foreign key of TA). The user does something in table TB. Following this change, you (your application) do some action to the parent(!) record in TA. If you want to add this action (done to a record in TA) also to the audit log, you use this function.

Example setting - you need to place the code in the correct functions! (maybe before_insert, after_insert, before_delete, after_delete):

TA has primary key field pkA. User is working in TB with parent record pkA=5 ($valueOfpkA)

Now the field TA.LogEdit ($fieldName) should be changed, once something in table TB happens.

Manual Installation (not recommended!)

Attention

Note 1

For the Search/Replace it's recommended to use 'Notepad++' available here: Notepad++ Home Page

Note 2

We suggest that you wait till your application is ready to go to production before making these changes - although this is NOT essential - (with the proviso that you BACK UP YOUR FILES FIRST!)

Note 3

When it comes to the tedious task of doing the Search/Replace in the Hooks folder, we suggest that you copy the hook files ONLY for the tables you wish to monitor into a separate directory and then BACKUP that directory. This way, you can do it speedily using Notepad++'s Search/Replace facility 'Find in Files' and do them all in just six shots.

Step 1. Extract the *auditlog* Zip and copy files

- `/hooks/auditLog_functions.php` : The functions that allow the audit log to work. Copy this into the /hooks folder of your application.
- `/hooks/auditLog_timeline.php` : This file is used to generate the timeline. Copy this into the /hooks folder of your application.
- `/admin/auditLog.php` : A table (filterable and pageable) that will be added to the Admin Menu Options. Copy this into the /admin folder of your application.

- `/admin/auditLogTimeLine.php` : This file will create the timeline from a link in the admin area. Copy this into the /admin folder of your application.

Step 2. Create the Audit Log Table using the *audit_tableSQL.sql* file provided.

You may want to adjust the auditor tablename before. See above: Custom table name for the auditor table (before installation).

Then just run the SQL with your favorite tool (https://www.phpmyadmin.net, https://www.adminer.org).

Step 3. Essential File Modifications

3.1 Include audit-base files: 'application_root/config.php'

Add the following to the bottom of the file.

```
if (session_status() == PHP_SESSION_NONE) { session_start(); }
$_SESSION['dbase'] = $dbDatabase;
if (!function_exists('table_before_change')) {
      $currDir = dirname(__FILE__);
      @require("$currDir/hooks/auditLog_functions.php");
}
```

3.2 Add page to the Admin Menu Options: 'admin/incHeader.php'

> Trick (as pictured and described in Advanced Audit Log Table (recommended), above):
> If you create a table in *AppGini* with the name as the Auditor table (i.e. Auditor) and the same fields (case sensitive) as the in the Auditor table, you can build a regular Audit-Table button from *AppGini* and let user access that with the regular permissions.
> Maybe you want to make sure, that no one can change anything in that table.
> The same goes for the timeline: If you give yourself (and others) access to the timeline as described Make the Auditor Timeline accessible to your users, you do not really need this adjustment.
> If you to this, you do not need to make changes in 'admin/incHeader.php' as described now (and thus Auditor will stay in your application even when regenerated).

Open `admin/incHeader.php` , search for:

```
<a class="navbar-brand" href="pageHome.php"><span class="text-
warning"><i class="glyphicon glyphicon-wrench"></i> Admin
Area</span></a>
```

and replace with:

```
<a class="navbar-brand" href="pageHome.php"><span class="text-
warning"><i class="glyphicon glyphicon-wrench"></i> Admin
Area</span></a><ul class="nav navbar-nav">
      <li class="dropdown">
            <a href="#" class="dropdown-toggle" data-
toggle="dropdown"><i class="glyphicon glyphicon-tasks"></i> Audit Log
<b class="caret"></b></a>
            <ul class="dropdown-menu">
                  <li><a href="auditLogTimeLine.php"><i
class="glyphicon menu-item-icon text-info glyphicon-calendar"></i>
Auditor Timeline</a></li>
                  <li><a href="auditLog.php"><i class="glyphicon menu-
item-icon text-info glyphicon-th"></i> Auditor Table</a></li>
            </ul>
      </li>
</ul>
```

Step 4. Essential /hooks/folder-files modification

After you've read Note 3, above**!**

In the *temp* folder that contains the files from the hooks-folder for *all the tables that you wish to monitor*, make the following changes to all these files. Recommended: Do 'find in files'. Code changes/additions are **`color coded like this`**.

A. Do 'find in files' for: (Remember to set the correct directory!)

```
init(&$options, $memberInfo, &$args){
```

and replace with:

```
init(&$options, $memberInfo, &$args){
$_SESSION ['tablenam'] = $options->TableName;
$_SESSION ['tableID'] = $options->PrimaryKey;
```

B. Do 'find in files' for:

```
after_insert($data, $memberInfo, &$args){
```

and replace with:

```
after_insert($data, $memberInfo, &$args){
table_after_change ($_SESSION, $memberInfo, $data, 'INSERTION');
```

C. Do 'find in files' for:

```
before_update(&$data, $memberInfo, &$args){
```

and replace with:

```
before_update(&$data, $memberInfo, &$args){
table_before_change($_SESSION, $data['selectedID']);
```

D. Do 'find in files' for:

```
after_update($data, $memberInfo, &$args){
```

and replace with:

```
after_update($data, $memberInfo, &$args){
table_after_change ($_SESSION, $memberInfo, $data, 'UPDATE');
```

E. Do 'find in files' for:

```
before_delete($selectedID, &$skipChecks, $memberInfo, &$args){
```

and replace with:

```
before_delete($selectedID, &$skipChecks, $memberInfo, &$args){
table_before_change($_SESSION, $selectedID);
```

F. Do 'find in files' for:

```
after_delete($selectedID, $memberInfo, &$args){
```

and replace with:

```
after_delete($selectedID, $memberInfo, &$args){
table_after_change ($_SESSION, $memberInfo, $selectedID, 'DELETION');
```

Remember to copy the files from the temp directory created in Note 3 back to the original Hooks folder!

AuditLog should now be working! Using this technique will allow you to keep other modifications made to the Hooks folder.

History / Versions / Changes

Adjustments by Olaf Nöhring, 2021-03 (https://datenbank-projekt.de) for v1.93:
- added/adjusted version numbers in all files
- created Clipboard_AppGini_AuditLog_Table.txt to easily copy the AppGini Table code.
- Adjusted documentation.

Adjustments by Olaf Nöhring, 2021-03 (https://datenbank-projekt.de) for v1.92:
- corrected docs after hint from Landinialejandro
- added hint that plugin needs to be run again after adding new tables in AG.

Adjustments by Olaf Nöhring, 2021-03 (https://datenbank-projekt.de) for v1.91:

- added the possibility to set the number of records for the Standard Audit Log Table (installed automatically)

Adjustments by Olaf Nöhring, 2021-03 (https://datenbank-projekt.de) for v1.9:
- added the new Timeline feature
- Suggested new menu in admin area to Landinialejandro (menu instead of direct link)

Adjustments by Olaf Nöhring, 2021-02 (https://datenbank-projekt.de) for v1.81:
- added protection from (un)wanted SQL: until now data was not cleaned before adding it to the auditor. Probably this could have been used as an attack vector.

Adjustments by Olaf Nöhring, 2021-02 (https://datenbank-projekt.de) for v1.78:
- changed SQL for auditor table. res_id should be (and is now) varchar to accommodate for non integer primary keys.
- Added res_id field fo internal auditLog table (which is not recommended to be used, see Advanced Audit Log Table (recommended))

Adjustments by Olaf Nöhring, 2021-02 (https://datenbank-projekt.de) for v1.77:
- corrected manual installation table_after_change call, usiing $selectedID instead of $data (which does not exist at this point)
- adjusted function table_after_change to accept $selectedID as direct variable (not array)
- corrected error when inserting a new record and calling table_before_change to document all values

Adjustments by Olaf Nöhring, 2021-02 (https://datenbank-projekt.de) for v1.76:
- Fixed and updated manual installation _init code
- added Fehler: Verweis nicht gefunden for easy, quick and correct creation of the auditor table in your AppGini application.
- cooperation with landinialejandro to make code cleaner for use in installation plugin

Adjustments by Olaf Nöhring, 2021-01 (https://datenbank-projekt.de) for v1.74:
- Added example screenshots of Audit Log view to docs
- Added hint how to deal with the need for exact position of audit-log calls

Adjustments by Olaf Nöhring, 2021-01 (https://datenbank-projekt.de) for v1.73:
- Adjusted docs for more clarity

Adjustments by Olaf Nöhring, 2021-01 (https://datenbank-projekt.de) for v1.72:
- Added link to docs for the wonderful plugin extension from landinialejandro which makes installation a walk in the park. Please see below
- Modified file structure of the zip files that holds the audit log to adjust for use in combination with the plugin.
- Changed formatting of the docs for better readability
- Restructured docs for plugin
- Added hint for auditor tablename to docs

Adjustments by Olaf Nöhring, 2021-01 (https://datenbank-projekt.de):

- removed bug when record is deleted (v1.71)
- added possibility to easily define the name of your auditor table in the beginning of the the files auditLog_functions.php and /admin/auditLog.php
- changed code in the way that now the order of fields in the database must not match the order of fields in your AppGini application anymore. In previous versions the order must be the same, otherwise it would mess up the logging. Now this problem should be solved.
- added a new function `Audit_Manually` which allows checking for changes on another table and documenting those (see description below for more information).
- transformed docs to PDF for easier editing
- changed audit_tableSQL.sql to make larger fields for table and fieldnames

Adjustments by Olaf Nöhring, 2019-12 (https://datenbank-projekt.de):
- improved INSERTION: Now, all non-empty fields are written to the auditor table after insert. Until now, only the new primary key was written.
- you can easily use a different table name for the auditor. Simply adjust
  $audittablename = "auditor";
  in function table_after_change in auditLog_functions.php (and the setup sql or course). - changed auditor table name from Auditor to auditor (script and setup). Note: On Linux systems the tablenames are case sensitive!

Adjustments by Olaf Nöhring, 2019-06 (https://datenbank-projekt.de):
- Trick: Added in docs. Remove access to Auditor from Admin menu, but use regular AppGini table instead, so Auditor stays even when application is regenerated.
- Trick: Remove changes from 'application_root/lib.php', instead place code in config.php which stays in place, even when the application is regenerated.
- Changes to auditLog_functions.php, added , $eo to SQL queries (following vaalonv tip)
  Instead of foreign keys you will see the values the user actually selected (old code to see FKs still in the file). For this to work correct, make sure the order of the fields in your database is exactly like the order of the fields in specific table in AppGini!