

AppGini Documentation

Contents

AppGini Online Guide	8
Introduction to AppGini	8
Key Features of AppGini	8
Supported Technologies	9
Getting Started	9
Install AppGini on Windows	10
Upgrading an existing installation	10
Uninstalling AppGini	14
Related topics	14
AppGini projects	15
What is an AppGini project?	15
How do I start a new project?	15
The project name	15
The project window	15
How do I save a project?	16
Auto save of projects	16
Opening an existing project	16
Advanced: The project file format	17
Getting help while you work	18
Project properties pane	19
Working with tables	20
How can I add a table to a project?	20
How can I rename a table?	20
How can I delete a table?	20
What about table properties?	21
Can I use a table from another AppGini project?	21
Anonymous table permissions	21
Table properties pane	22
Toolbar > New project	23
Table properties > Table view title	23
Toolbar > Open Project	23
Toolbar > Save Project	23
Toolbar > Project Properties	23
Toolbar > Application Theme	23
Toolbar > New Table	24
Toolbar > New Field	24
Toolbar > Delete Selected Table/Field	24
Toolbar > Move Table/Field up	24
Toolbar > Move Table/Field down	24
Toolbar > Generate PHP Code	24
Toolbar > Preferences	24

Toolbar > Help	24
Toolbar > About AppGini	24
Toolbar > Ideas and tips on our Twitter page	25
Toolbar > Learn to use Appgini through our YouTube playlist	25
Toolbar > Exit	25
Project Browser Window	25
Table properties > Table view title	25
Table properties > Description	25
Table properties > Records per page	25
Table properties > Show quick search box	25
Table properties > Default sort by,Descendingly	26
Table properties > Allow sorting	26
Table properties > Allow filters	26
Table properties > Allow saving data to CSV files	26
Table properties > Allow print-view	26
Table properties > Allow users to save filters	26
Table properties > Hide link in homepage	26
Table properties > Allow mass delete	26
Table properties > Filter before showing table view	27
Table properties > Hide link in navigation menu	27
Table properties > Show record count in Homepage	27
Table properties > Table group	27
Table properties > Table template	27
Table properties > Detail view title	27
Table properties > Redirect after insert	27
Table properties > Display a link to children records from	27
Table properties > Default focus field	28
Table properties > Enable detail view	28
Table properties > Allow detail print-view	28
Table properties > Hide 'Save As Copy' when editing	28
Table properties > Allow adding new records from Homepage	28
Table properties > Delete records even if they have children records	28
Table properties > Display detail view in a separate page	28
Table properties > Keep action buttons visible while scrolling down	29
Table properties > Table technical documentation	29
Table properties > Edit Technical Documentation	29
Table properties > Technical documentation preview	29
Table properties > Parent/Children settings	29
Table properties > Table icon	29
Search box	29
File menu	29
Tables menu	30
Project menu	30
Add-ons menu	30
Help menu	30
Copy	30
Paste	30
Toggle Highlight	30
Activate/deactivate help for toolbar icons	30
Working with table fields	31
How can I add a field to a project?	31
How can I rename a field?	31
Can I change the order of fields in a table?	31
Can I clone/copy a field?	32
How can I delete a field?	32
What about field properties?	32

Field properties pane	33
Caption	33
The Media Tab	34
Link option	34
The Image option	34
The File upload option	34
Google Maps	37
YouTube video	37
Understanding lookup fields	41
How will a lookup field appear in the generated application?	41
Displaying lookup fields as an options list (radio buttons)	43
Related screencasts	44
AppGini lookup fields and master detail pages	44
Using auto-fill look-up fields to automatically populate fields from another table	44
Creating cascading drop downs with AppGini	44
Displaying child info (count + add new) in the table view	44
Calculated fields	47
What are calculated fields?	47
Conditions for a field to become a calculated field	47
How to configure a calculated field	47
Special variables for use in calculated field queries	49
The query helper	50
Debugging your query	51
Batch-updating calculated fields via command line	51
Basic examples of calculated fields	52
Calculate subtotal for an invoice line by multiplying unit price and quantity	52
Automatic code by concatenating 2 or more fields	52
More advanced examples of calculated fields	52
Updating batch status to 'Consumable', 'Warning' or 'Expired' based on expiry date	52
Invoice subtotal by summing subtotals of invoice items	53
Looking for more help with queries?	53
Known issues	53
Working with styles	55
Preview in web browser	55
Generating the PHP application	56
Keyboard shortcuts in AppGini	60
Automatic application uploader	61
Smart features of the automatic file uploader	61
How to enable automatic file uploading	61
How to use automatic file uploading	62
Troubleshooting	63
Security considerations	65
Setting the child record owner to match the owner of its parent record	68
Updating the owner of existing child records	69
A practical example: Set the country sales manager as the owner of all orders and order items of that country	69
Working with the generated web database application	73
A briefing of the generated files	74
Setup	76
STEP 1 OF 3: Running the setup script	76

STEP 2 OF 3: Logging to the admin control panel	78
STEP 3 OF 3: Changing the admin password	78
Working with tables and records	80
The application homepage	80
The Table View page	80
The Detail View	82
Working with filters	83
Related screencasts	84
The admin interface	85
Member groups	85
Sample scenario: A content publishing application	85
Accessing the admin homepage	85
Managing groups	85
Managing members	88
Managing records	89
Other features of the admin interface	90
Shortcut keys	91
LDAP Authentication	93
Video overview of LDAP settings in AppGini apps	93
Enabling LDAP Extension in PHP	93
Configuring LDAP Settings	93
Specifying the LDAP Server	94
LDAP Version	94
User DN (Distinguished Name) Pattern	94
Handling Non-Existent Users	94
Important Considerations	95
Testing LDAP Integration	95
Troubleshooting	95
Conclusion	95
Enabling LDAP Extension in PHP	96
For Linux (Debian/Ubuntu)	96
For Linux (CentOS/RHEL)	96
For Windows	96
Verifying LDAP Extension Activation	96
Advanced topics	98
Hooks (AKA events)	99
How do hooks work?	99
Global hooks	100
login_ok()	100
Parameters:	100
Return value:	100
Example:	100
login_failed()	101
Parameters:	101
Return value:	101
Example:	101
member_activity()	101
Parameters:	102
Return value:	102
Example:	102
sendmail_handler()	103

Parameters:	103
Return value:	103
<code>child_records_config()</code>	103
Parameters:	103
Return value:	104
Table-specific hooks	105
<code>tablename_init()</code>	105
Parameters	105
Return value	105
Example	105
<code>tablename_header()</code>	106
Parameters	106
Return value	107
Example	107
<code>tablename_footer()</code>	107
Parameters	108
Return value	108
Example	108
<code>tablename_before_insert()</code>	108
Parameters	109
Return value	109
Example 1	109
Example 2 (AppGini 5.90+)	109
<code>tablename_after_insert()</code>	109
Parameters	110
Return value	110
Example 1	110
Example 2	110
<code>tablename_before_update()</code>	110
Parameters	111
Return value	111
Example	111
Another example	111
<code>tablename_after_update()</code>	112
Parameters	112
Return value	112
Example	112
<code>tablename_before_delete()</code>	112
Parameters	112
Return value	112
Example	113
<code>tablename_after_delete()</code>	113
Parameters	113
Return value	113
Example	113
<code>tablename_dv()</code>	114
Parameters	114
Return value	114
Example	114
<code>tablename_csv()</code>	114
Parameters	115
Return value	115
Example	115
Using lookup fields in calculations	116
Products table	116
Order Items table	116

Order Items table joined with Products table	117
Adding custom “batch actions” that apply to multiple records	119
Adding custom batch actions	119
Example: Adding a batch action to print mailing labels for selected records	120
DataList object	126
Here is a list of the editable properties of the DataList object	126
AllowCSV	126
AllowDeleteOfParents	126
AllowFilters	126
AllowPrinting	126
AllowSavingFilters	126
AllowSorting	126
CSVSeparator	126
ColCaption	126
ColNumber	127
ColWidth	127
DefaultSortDirection	127
DefaultSortField	127
DVClasses	127
FilterPage	127
PrimaryKey	127
QueryFieldsCSV	127
QueryFieldsFilters	127
QueryFieldsTV	127
QueryFrom	127
QuickSearch	128
QuickSearchText	128
RecordsPerPage	128
RedirectAfterInsert	128
SelectedTemplate	128
SeparateDV	128
ShowTableHeader	128
TableTitle	128
Template	128
TemplateDV	128
TemplateDVP	128
TVClasses	129
Inspecting the DataList object	129
memberInfo array	130
Magic files in the hooks folder	131
tablename-dv.js: modifying the behavior of the detail view through JavaScript	131
tablename-tv.js: modifying the behavior of a specific table through JavaScript	131
tablename.fieldname.csv: changing the contents of options lists	132
WindowMessages class	133
How does it work?	133
Including the window ID when redirecting to another page	133
Including the window ID in a form in custom pages	133
How to display the messages?	134
Table and detail view classes	135
Adding custom limited-access pages and reports	137
Control access to your custom page	137

Third party resources used by AppGini applications	140
Command-line parameters	141
Special links and URL parameters	142
Here is a table listing common URL parameters and their usage	143
SortField	143
SortDirection	143
FilterAnd[x]	143
FilterField[x]	143
FilterOperator[x]	143
FilterValue[x]	143
SearchString	143
FirstRecord	143
Print_x	143
addNew_x	144
filterer_{fieldname}	144
SelectedID	144
dvprint_x	144
delete_x	144
noQuickSearchFocus	144
From Data to Dashboards, A Guide to Redash Integration with AppGini	145
What is Redash?	145
Installing Redash on your server	145
Connecting Redash to your AppGini application	145
Here is a screencast showing the above steps	147
Creating a query in Redash	147
Here is a screencast showing the above steps	147
Creating a dashboard in Redash	148
Creating alerts in Redash	148
Dive deeper into Redash	149
Performance considerations when using Redash	149
Conclusion	149
I see an error or a blank page in my AppGini app - how to troubleshoot?	150
Useful links	153
Setting up a test environment for your web applications	153
Uploading the generated code to your server	153
Our online course on customizing AppGini applications	153
Hosting providers	153
Code management	153
Reference material	153
Contribute to AppGini documentation	155

AppGini Online Guide

Introduction to AppGini

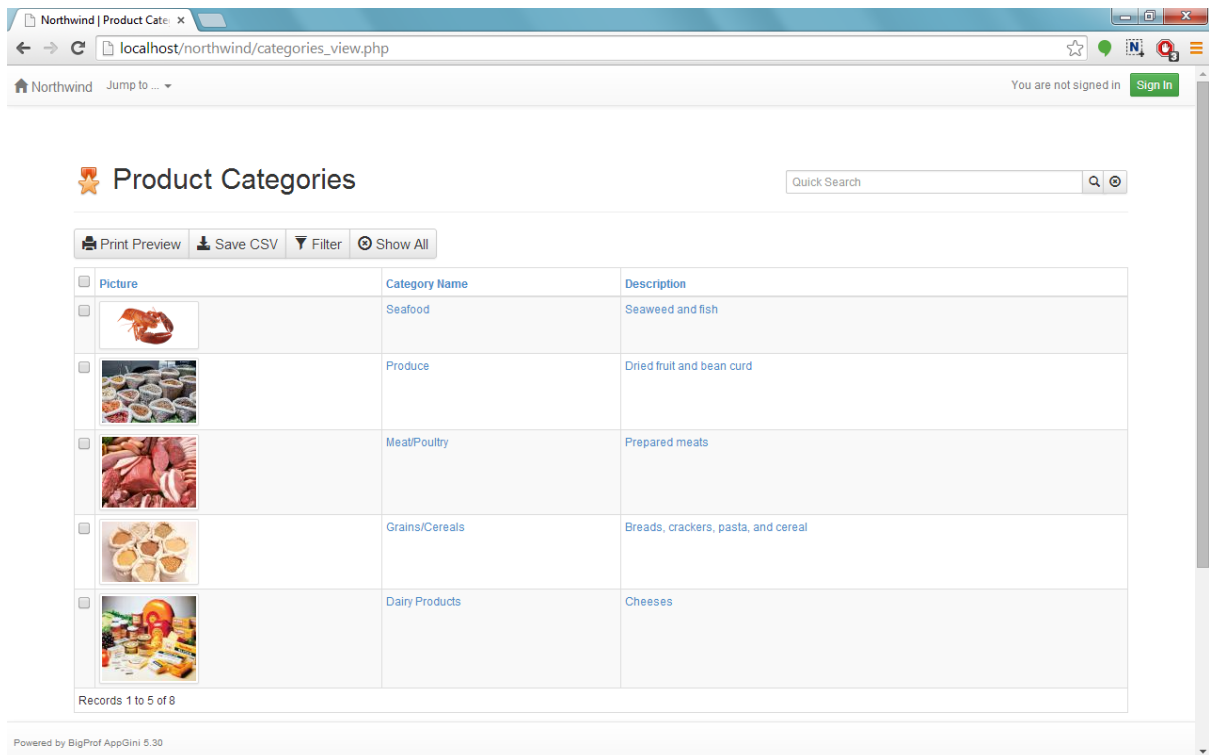


Figure 1: A web application interface generated by AppGini

AppGini is a powerful tool designed to simplify the creation of web-based database applications. Even without any programming skills, you can quickly design and generate fully functional web interfaces for your business applications. Simply define your data structure, configure a few settings, and with a click of the 'Generate' button, your application is ready!

Key Features of AppGini

AppGini equips users with a comprehensive set of tools to build professional-grade web database applications. These applications are ready to be deployed on any website and include a wide range of functionalities:

- **Data Management:** Effortlessly navigate through data, with capabilities to sort, filter, edit, insert, and delete records.
- **Data Exchange:** Provides options for data import and export, facilitating easy data migration and backup.
- **User Access Control:** Manage user and group permissions to ensure secure access to the database.
- **Enhanced Interactivity:** Incorporates foreign key support to create lookup fields, enhancing data integrity and usability.

- **Customization:** Offers extensive customization options for the application's appearance and behavior, adapting to specific needs.
- **Media Handling:** Supports the uploading of images and files, as well as Google maps, Youtube videos, and rich text fields, enriching the data records with multimedia content.

Supported Technologies

AppGini applications are generated in PHP and are designed to connect with MySQL/MariaDB databases. This selection leverages the widespread support and compatibility of PHP and MySQL/MariaDB across different hosting environments, ensuring that the applications you create are robust and deployable on virtually any server without requiring prior configuration.

Getting Started

Dive into the world of AppGini without needing to learn the intricacies of PHP or database management. The intuitive interface and comprehensive feature set allow you to start creating immediately. Whether you choose to explore the documentation further or jump straight into using the software, AppGini promises a smooth and productive experience.

Explore the potential of AppGini and start building your web database applications today. Join us as we uncover the powerful capabilities and ease of use that AppGini offers!

Install AppGini on Windows

After downloading AppGini (the trial or pro version), you can install it by following the steps below:

1. Double-click the downloaded setup file.

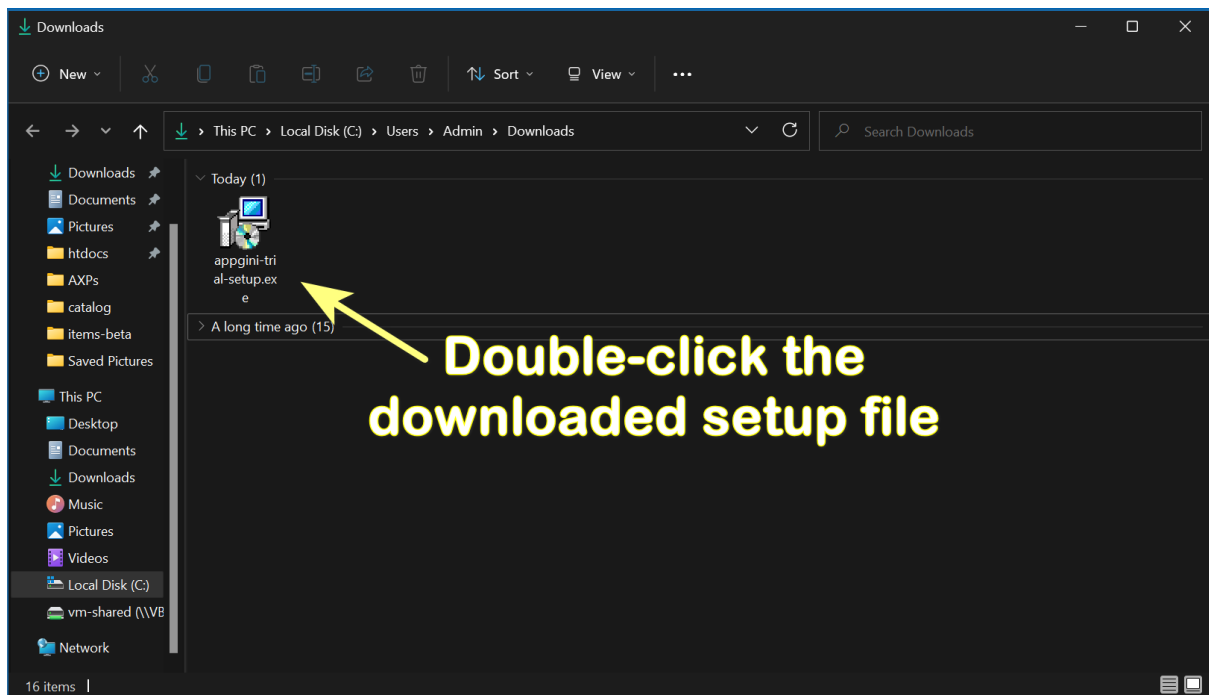


Figure 2: Double-click the downloaded setup file

2. Click 'Yes' if prompted by User Account Control
3. Click 'Next' to start the installation, and follow the steps in the setup wizard.
Setup will take a few seconds to prepare the installation.
4. Click 'Finish' to complete the installation.
5. AppGini is now installed on your computer. You can start it by double-clicking the desktop shortcut or by clicking the start menu shortcut.
If you can't see the shortcut in the start menu, you can type **AppGini** in the search box to find it.
6. The first time you run AppGini, you'll be asked to choose how you want to begin your work. You can choose to create a new project, open an existing project, among other options.

Upgrading an existing installation

If you're upgrading from a previous version of AppGini, you can install the new version over the old one. Your existing projects and settings will be preserved. You can follow the same steps above to install the new version. Just make sure to close AppGini before starting the installation.

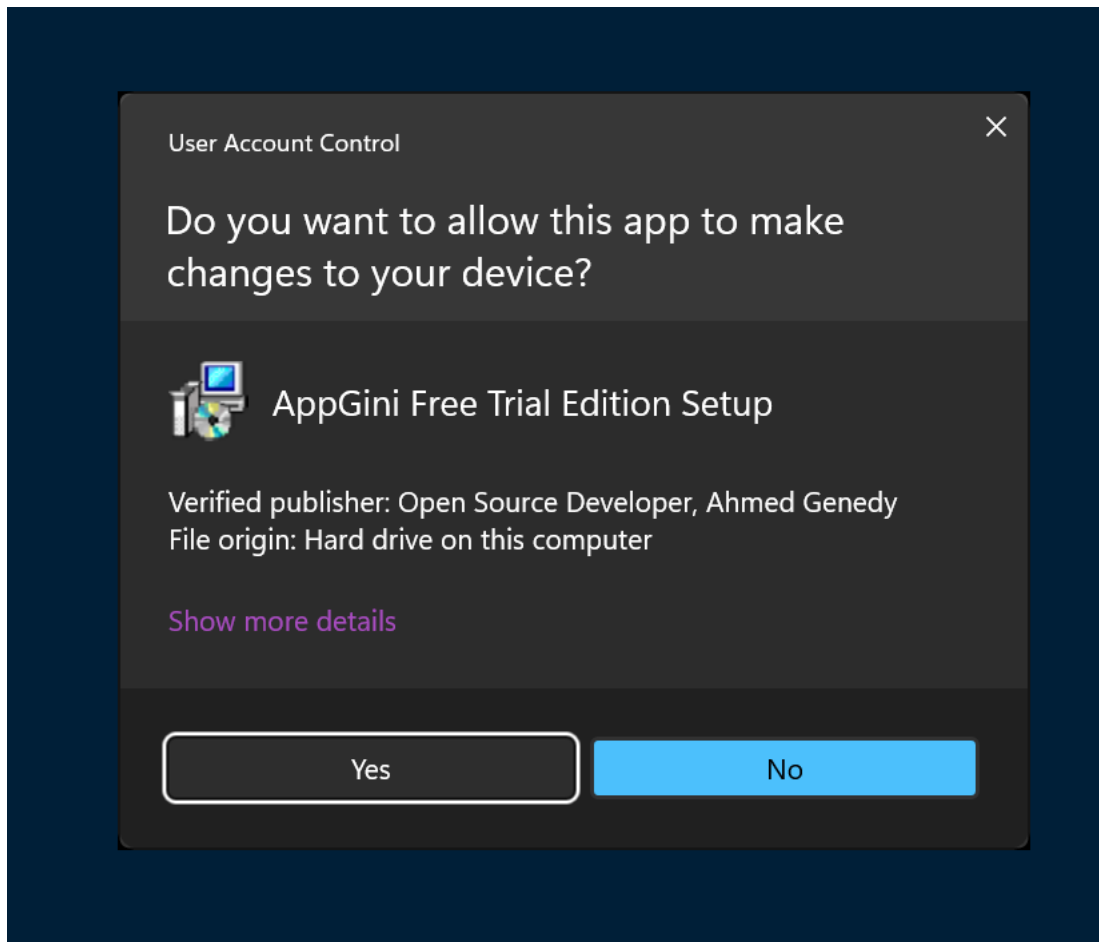


Figure 3: Click 'Yes' when prompted by User Account Control

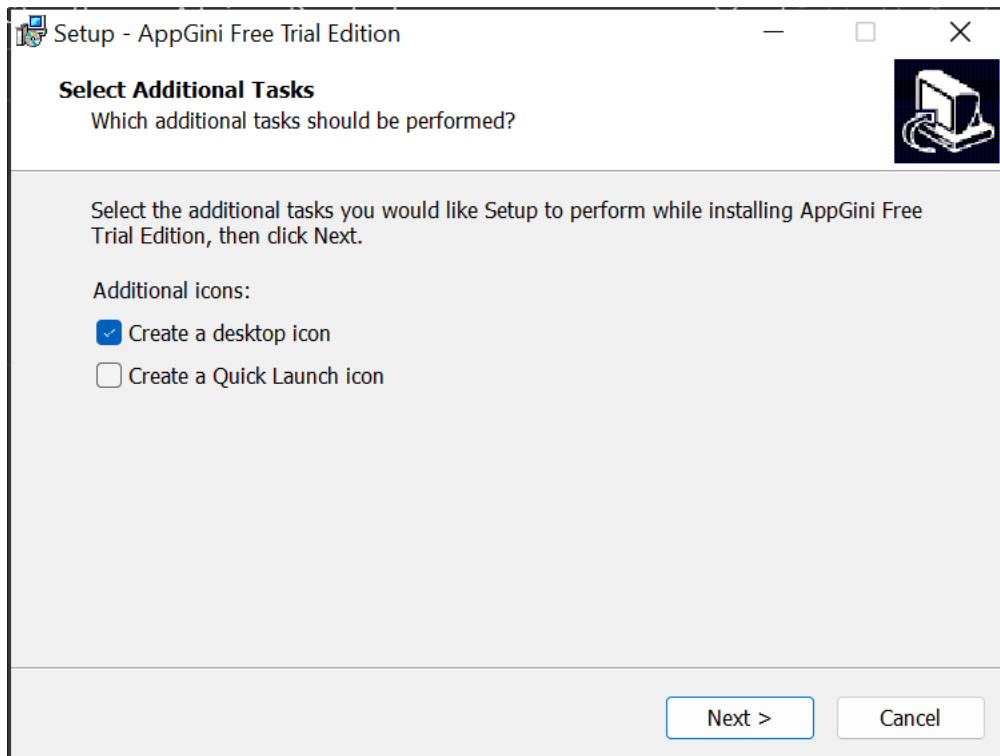


Figure 4: Click 'Next' to start the installation

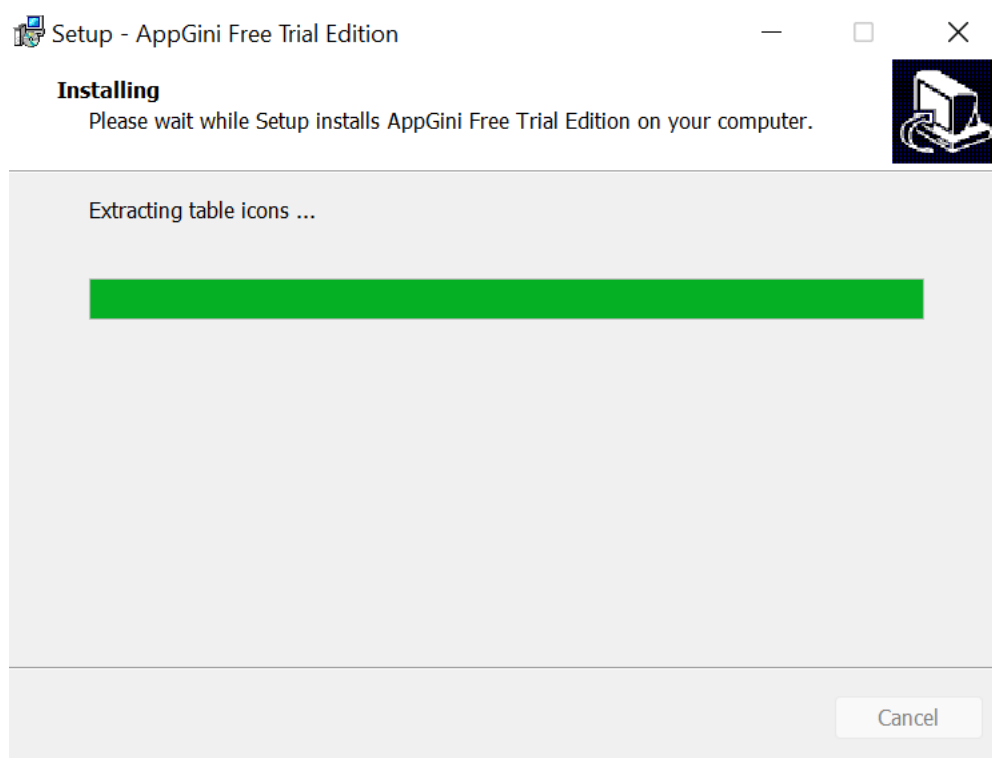


Figure 5: Setup progress

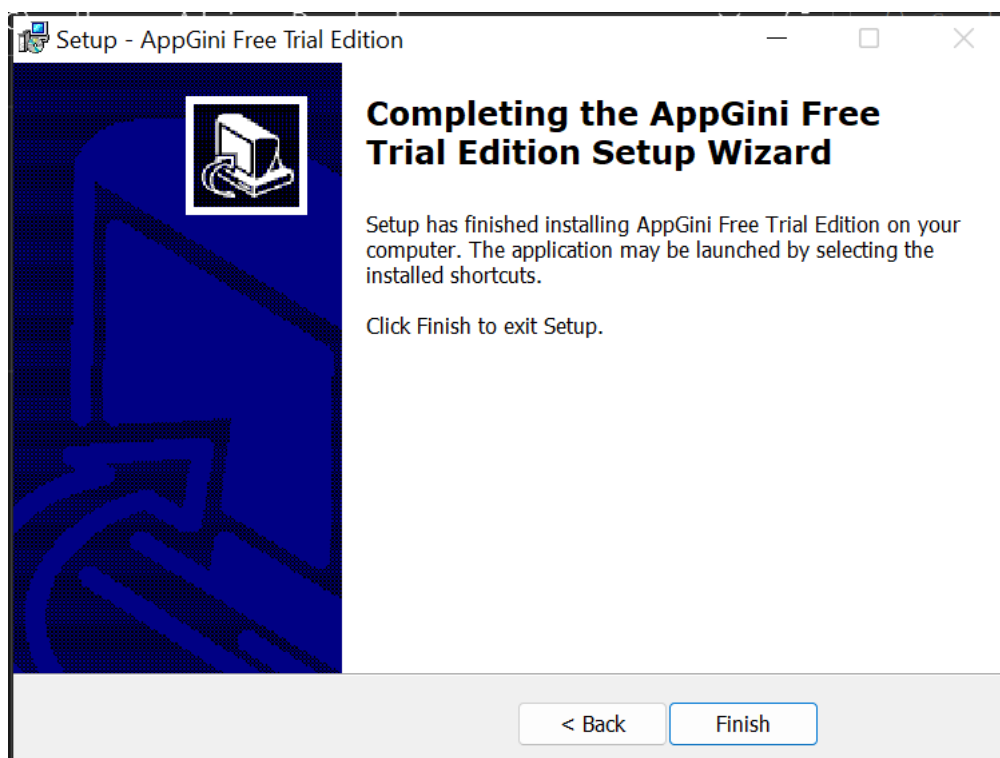


Figure 6: Click 'Finish' to complete the installation

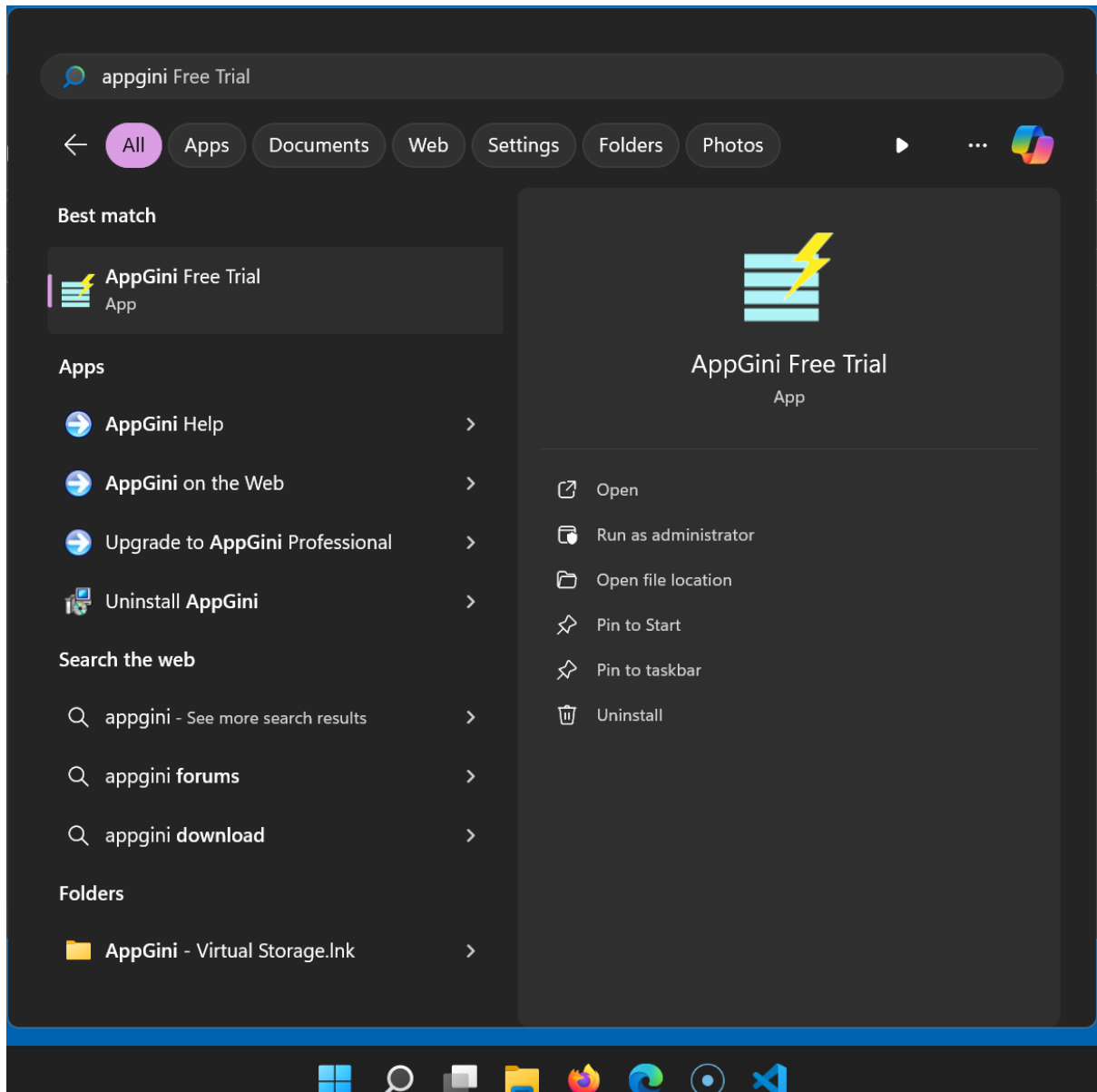


Figure 7: Start menu shortcut

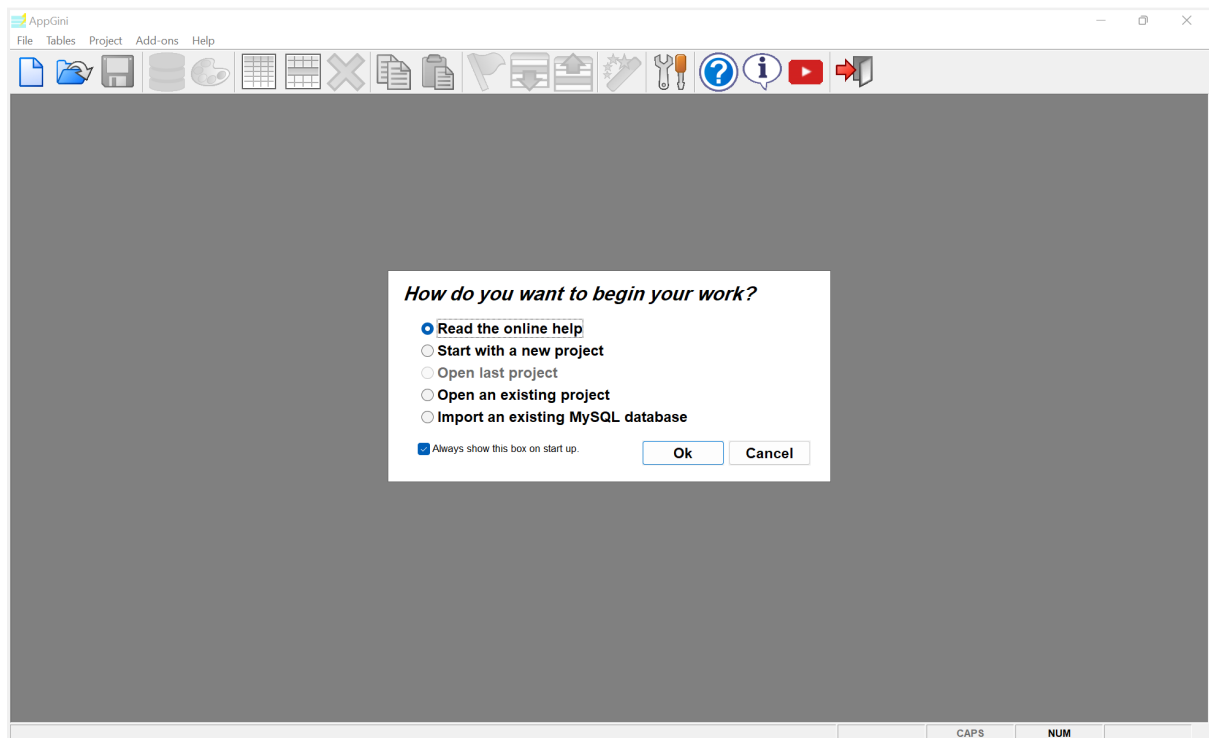


Figure 8: How do you want to begin your work?

Uninstalling AppGini

To uninstall AppGini, you can use the ‘Add or remove programs’ feature in Windows. Here’s how:

1. Open the ‘Add or remove programs’ window by typing ‘Add or remove programs’ in the start menu search box and clicking the result.
2. Find ‘AppGini’ in the list of installed programs, click it, and then click ‘Uninstall’.
3. Follow the steps in the uninstall wizard to remove AppGini from your computer.

After uninstalling AppGini, you can delete the installation folder to remove any remaining files.

Related topics

- Installing AppGini on MacOS or Linux
- Installing AppGini on a remotely-hosted Windows VM

AppGini projects

What is an AppGini project?

To create a web database application using AppGini, you start by creating a project. In the project, you define your database tables and fields, and configure your application's appearance and behavior. Once you have finished working on your project, you can generate the PHP code for your application by clicking the “Generate” button on the toolbar or pressing F5. AppGini will then save the generated code to a folder of your choice. Finally, you can deploy the generated files to your web server.



Figure 9: The Generate button on the toolbar.

How do I start a new project?

To start a new project in AppGini, follow these steps:

1. Launch AppGini.
2. From the File menu, select New, or click the ‘New Project’ icon on the toolbar.
3. A new project window will appear, as shown below.
4. By default, the project is named ‘new_db’. To change the name, click on the project name at the top of the left section (the project browser pane), then press **F2** and type the new name.

The project name

Note that the project name you specify in AppGini doesn't have to match the name of the database on your server. It's used only for display purposes and will be used as the basis for the application title, which is displayed at the top of the project properties pane (see the screenshot below) and in the generated application's title bar.

You can configure the actual database name and credentials during the generated application setup, which is explained in detail [here](#).

The project window

This is your working area. This window has two sections: the project browser pane at the left lets you view your project components (the project, tables and fields) in an easy to navigate hierarchical manner. The right section is the properties pane. When you click on any item in the project browser, its properties are displayed in the properties pane.

At the bottom of the project window, you'll find the project search box. You can use this box to quickly find any item in your project by typing a few characters of its name.

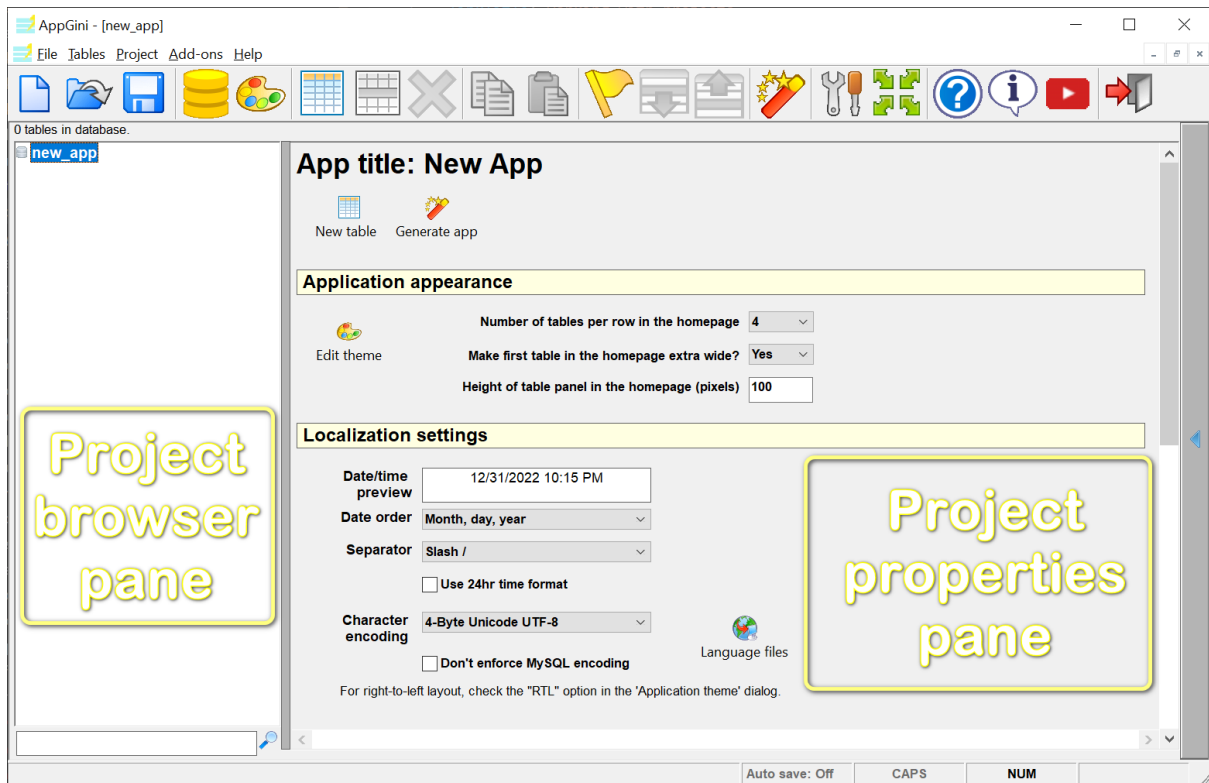


Figure 10: New project window

How do I save a project?

In the professional version of AppGini, projects can be saved as project files, with the extension `.exp`. You can open them later to continue working on your project or modify it. The freeware version can open project files but can not save changes to them.

To save a project, click the 'Save' icon on the toolbar or press **Ctrl + S**. You'll be prompted to choose a location to save your project file if you haven't saved it before. You can also save your project by selecting 'Save' from the File menu.

Note: Saving a project file doesn't save the generated application files. To save the generated application files, you need to click the 'Generate' button on the toolbar or press **F5**.

Auto save of projects

You can configure AppGini to automatically save your project every few minutes. To do this, click the 'AppGini Preferences' icon on the toolbar, then check the 'Auto-save interval (minutes)' option and specify the interval in minutes.

Opening an existing project

When you launch AppGini, you'll see the 'How do you want to begin?' dialog. You can choose to start a new project, open an existing project, or open last project. If you don't see this dialog, it means you've disabled it before. You can enable it again by checking the 'Show how do you want to begin' option in the 'AppGini Preferences' dialog.

You can also open an existing project by selecting 'Open' from the File menu or clicking the 'Open Project' icon on the toolbar. Then, navigate to the location of your project file and select it.

Advanced: The project file format

AppGini project files have the extension `.axp`. They are XML files that store all the information about your project, including the tables, fields, and their properties, as well as the project settings. You can open an `.axp` file in a text/code editor to see its contents. You can also make modifications to the file, but be careful not to corrupt it, as this may prevent you from opening the project in AppGini.

Getting help while you work

In addition to this online help file, there are several additional help resources: there is the continuously expanding tips and tutorials section, AppGini FAQs, the context help inside AppGini, and the AppGini community forums. Context help is a handy tool for obtaining help while you work with your projects without having to be online.



Figure 11: F1

To activate context help, press F1. This would open the help section at the right side of the AppGini window. This section displays help on whatever element has the focus at the moment. You can use the mouse or the Tab key to move the focus between all the elements of the project window. To hide the help section, press F1 again.

Project properties pane

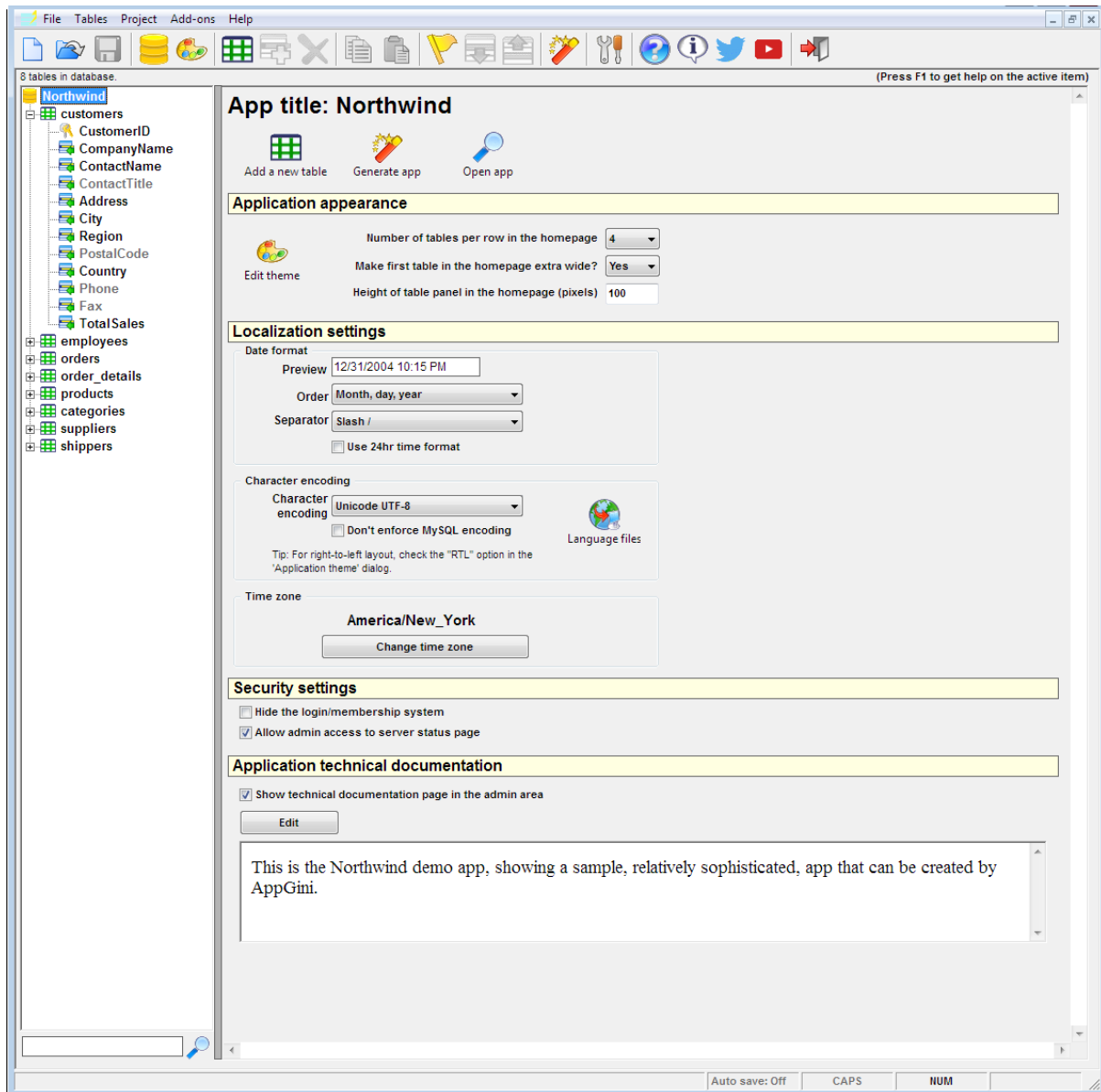


Figure 12: Project properties pane

TODO: Add description of each property in the project properties pane.

Working with tables

How can I add a table to a project?

Make sure you have an open project, and click on the 'New Table' icon on the tool bar, or from the Tables menu, select New Table.

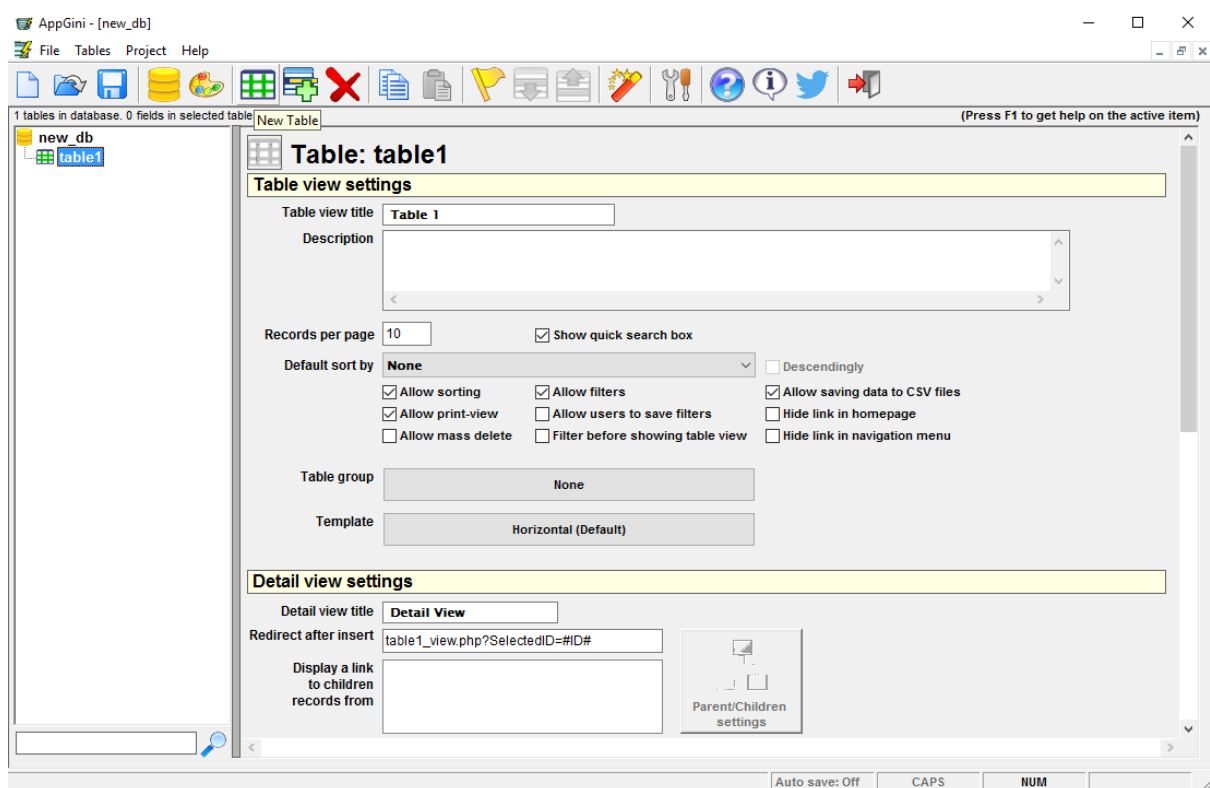


Figure 13: New Table

How can I rename a table?

Select the table by clicking on it in the Project Browser (if it is not already highlighted) and click on its name or press F2. A cursor will appear allowing you to rename the table.

How can I delete a table?

Select the table by clicking on it in the Project Browser and press the Delete button. Please note that deleting a table from AppGini does not delete it from your database if it already exists. This is done to protect your data from getting deleted unintentionally. If you really want to delete (drop) the table from your database, you can do so using phpMyAdmin or a similar MySQL administration utility.

What about table properties?

Click with the mouse on any property and press F1 to obtain help about its function.

Can I use a table from another AppGini project?

Yes, instead of having to recreate a table and all its fields, you can simply copy it from another AppGini project and paste it into your current project. Just use the Copy and Paste icons from the top toolbar.

Anonymous table permissions

By default, your new table will be accessible only to the admin users after uploading your AppGini application to the server. You can change the table permissions to allow access for anonymous users and/or other groups from the admin area of the generated application.

Table properties pane

```

  <area shape="rect" href="#file" coords="35,2,60,18" />
  <area shape="rect" href="#tables" coords="67,2,107,18" />
  <area shape="rect" href="#project" coords="114,2,158,18" />
  <area shape="rect" href="#add-ons" coords="165,2,218,18" />
  <area shape="rect" href="#help1" coords="226,2,257,18" />
  <area shape="rect" href="#new-project" coords="12,23,42,58" />
  <area shape="rect" href="#open-project" coords="46,23,82,58" />
  <area shape="rect" href="#save" coords="85,23,120,58" />
  <area shape="rect" href="#project-properties" coords="134,23,169,58" />
  <area shape="rect" href="#application-theme" coords="172,23,209,58" />
  <area shape="rect" href="#new-table" coords="220,23,255,58" />
  <area shape="rect" href="#new-field" coords="260,23,294,58" />
  <area shape="rect" href="#delete" coords="297,23,334,58" />
  <area shape="rect" href="#copy-table-or-field" coords="346,23,378,58" />
  <area shape="rect" href="#paste-table-or-field" coords="386,23,417,58" />
  <area shape="rect" href="#toggle-highlight" coords="431,23,466,58" />
  <area shape="rect" href="#move-table-or-field-down" coords="470,23,505,58" />
  <area shape="rect" href="#move-table-or-field-up" coords="509,23,544,58" />
  <area shape="rect" href="#generate-php-code" coords="557,23,592,58" />
  <area shape="rect" href="#appgini-preferences" coords="605,23,637,58" />
  <area shape="rect" href="#help2" coords="650,23,685,58" />
  <area shape="rect" href="#about-appgini" coords="689,23,724,58" />
  <area shape="rect" href="#get-great-ideas-and-tips-on-our-twitter-page" coords="729,23,762,58" />
  <area shape="rect" href="#learn-to-use-appgini-thru-our-youtube-playlist" coords="768,23,801,58" />
  <area shape="rect" href="#exit" coords="815,23,850,58" />
  <area shape="rect" href="#created-tables-and-fields" coords="14,79,212,910" />
  <area shape="rect" href="#table-icon" coords="227,86,260,119" />
  <area shape="rect" href="#table-view-settings" coords="226,122,1085,142" />
  <area shape="rect" href="#table-view-title" coords="255,149,564,170" />
  <area shape="rect" href="#description" coords="274,173,995,250" />
  <area shape="rect" href="#records-per-page" coords="237,257,397,281" />
  <area shape="rect" href="#show-quick-search-box" coords="488,261,639,280" />
  <area shape="rect" href="#default-sort-by-desendingly" coords="259,284,901,310" />
  <area shape="rect" href="#allow-sorting" coords="345,313,474,331" />
  <area shape="rect" href="#allow-filters" coords="485,313,691,331" />
  <area shape="rect" href="#allow-saving-data-to-csv-files" coords="697,313,901,331" />
  <area shape="rect" href="#allow-print-view" coords="345,334,474,351" />
  <area shape="rect" href="#allow-users-to-save-filters" coords="485,334,691,351" />
  <area shape="rect" href="#hide-link-in-homepage" coords="697,334,901,351" />
  <area shape="rect" href="#allow-mass-delete" coords="345,355,474,371" />
  <area shape="rect" href="#filter-before-showing-table-view" coords="485,355,691,371" />
  <area shape="rect" href="#hide-link-in-navigation-menu" coords="697,355,901,371" />
  <area shape="rect" href="#show-record-count-in-homepage" coords="697,376,901,394" />
  <area shape="rect" href="#table-group" coords="270,412,699,450" />
```

```

<area shape="rect" href="#table-template" coords="270,454,699,491" />
<area shape="rect" href="#detail-view-settings" coords="226,508,1085,532" />
<area shape="rect" href="#detail-view-title" coords="253,536,611,560" />
<area shape="rect" href="#redirect-after-insert" coords="227,562,611,587" />
<area shape="rect" href="#display-a-link-to-children-records-from" coords="263,590,611,661" />
<area shape="rect" href="#default-focus-field" coords="621,536,1062,561" />
<area shape="rect" href="#parent-and-children-setting" coords="621,562,841,660" />
<area shape="rect" href="#enable-detail-view" coords="349,666,611,682" />
<area shape="rect" href="#allow-print-view" coords="349,685,611,702" />
<area shape="rect" href="#hide-save-as-copy-when-editing-records" coords="349,706,611,723" />
<area shape="rect" href="#allow-adding-new-records-from-homepage" coords="349,727,611,744" />
<area shape="rect" href="#delete-records-even-if-they-have-children-records" coords="620,666,915,682" />
<area shape="rect" href="#display-detail-view-in-a-separate-page" coords="620,685,915,702" />
<area shape="rect" href="#keep-action-buttons-visible-while-scrolling-down" coords="620,706,915,723" />
<area shape="rect" href="#appgini-search-box" coords="12,962,210,990" />
<area shape="rect" href="#table-technical-documentation" coords="226,758,1085,784" />
<area shape="rect" href="#edit-technical-documentation" coords="236,791,347,819" />
<area shape="rect" href="#technical-documentation-preview" coords="236,822,1077,931" />
</map>

```

Toolbar > New project

Click on this icon to start a new empty project. This is the starting point for any AppGini application. You will have an empty project that contains only an empty database and predefined default styles.

[back to top](#)

Table properties > Table view title

The table view title is the title of the table as it will appear in the pages of the generated PHP application.

[back to top](#)

Toolbar > Open Project

Click on this icon to open a previously saved project.

[back to top](#)

Toolbar > Save Project

Click on this icon to save the currently open project. If you haven't saved your project before, you'll be prompted for a file name and location.

[back to top](#)

Toolbar > Project Properties

Click on this icon to bring the Project Properties area where you can specify the database connection parameters.

[back to top](#)

Toolbar > Application Theme

Click on this icon to preview (and edit) the theme of your application.

[back to top](#)

Toolbar > New Table

Click on this icon to add a new table to your database.

[back to top](#)

Toolbar > New Field

Click on this icon to add a new field to the current table.

[back to top](#)

Toolbar > Delete Selected Table/Field

Click on this icon to remove the highlighted field or table from your database.

[back to top](#)

Toolbar > Move Table/Field up

Click on this icon to move the highlighted field down/up in the current table.

[back to top](#)

Toolbar > Move Table/Field down

Click on this icon to move the highlighted field down/up in the current table.

[back to top](#)

Toolbar > Generate PHP Code

Click on this icon to let AppGini generate the PHP code for your web database application based on the database, tables and fields you defined in your project. You'll be prompted for a location to save the generated files.

[back to top](#)

Toolbar > Preferences

Click on this icon to open the preferences window, where you can set various AppGini options.

[back to top](#)

Toolbar > Help

Click on this icon to view the online help.

[back to top](#)

Toolbar > About AppGini

Click on this icon to view version and contact information for AppGini

[back to top](#)

Toolbar > Ideas and tips on our Twitter page

Click on this icon to visit (and optionally subscribe to) our Twitter feed, which includes a lot of helpful resources and tips to get the most out of AppGini.

[back to top](#)

Toolbar > Learn to use Appgini through our YouTube playlist

Click on this icon to visit (and optionally subscribe to) our Youtube channel where you can watch many short video tutorials explaining various features of AppGini.

[back to top](#)

Toolbar > Exit

Click on this icon to quit the program. If you have an open project, you'll be asked to save your work first.

[back to top](#)

Project Browser Window

This is the project browser window where you can see a list of the database, table and field names in the current project arranged in a hierarchical view. Click on any item on the window to view and edit its properties in the Properties window.

[back to top](#)

Table properties > Table view title

The table view title is the title of the table as it will appear in the pages of the generated PHP application.

[back to top](#)

Table properties > Description

If you provide a description for the table, this description will be displayed in the home page of the generated application. You can use HTML code in this description.

[back to top](#)

Table properties > Records per page

The number of records that appear on one page in the generated PHP application. Records are displayed as horizontal rows of a table. If you put a large number here, your application user might have to scroll down the page to view records.

[back to top](#)

Table properties > Show quick search box

If you want to display a quick search box above the table view, check this option. When the user enters a word in that search box and clicks the search button or presses Enter key, the table is searched for matches in any field. This is much easier than the advanced filters if you want a simple search technique.

[back to top](#)

Table properties > Default sort by, Descendingly

If you'd like records in the table view sorted by default, select the field that you want to sort by from the 'Default sort by' drop down. The default sorting direction is ascendingly (A-Z, 0-9) unless you check the 'Descendingly' option. If you don't want any default sorting, select 'None' from the 'Default sort by' drop down.

[back to top](#)

Table properties > Allow sorting

This option controls whether users are allowed to sort records in the table or not. Note that this doesn't affect the table in the database, only the query that displays data to the user.

[back to top](#)

Table properties > Allow filters

This option controls whether users are allowed to filter records in the table or not. Note that this doesn't affect the table in the database, only the query that displays data to the user.

[back to top](#)

Table properties > Allow saving data to CSV files

If you check this option, users will see a Save button above the table view that allows them to save data as a CSV file (Comma-Separated values).

[back to top](#)

Table properties > Allow print-view

This option controls whether users are allowed to view the table data as a printer-friendly page.

[back to top](#)

Table properties > Allow users to save filters

If you check this option, users will see a Save button in the filters page. If they click this Save button, they will view some HTML code that they can copy and paste to any external web page. This HTML code creates a button linking users to the filtered table view (without having to redefine filters). TIP: Usually, you would want to allow this option temporarily till all users have determined which preset filters they want to save. Later on, you can disable this option and regenerate the code.

[back to top](#)

Table properties > Hide link in homepage

If checked, this table will not have a link to it in the homepage of the generated application. Usually used with the option 'Hide link in navigation menu' checked as well. This is useful for example if this is a child table and you want users to access it only from its parent table rather than directly through a link.

[back to top](#)

Table properties > Allow mass delete

This option controls whether users who have permission to delete records can delete multiple records at once.

[back to top](#)

Table properties > Filter before showing table view

When this option is checked, users see the filters page first before seeing the table data. This is useful if you want users to search the table and display the search results to them.

[back to top](#)

Table properties > Hide link in navigation menu

If checked, this table will not have a link to it in the navigation menu of the generated application. Usually used with the option 'Hide link in homepage' checked as well. This is useful for example if this is a child table and you want users to access it only from its parent table rather than directly through a link.

[back to top](#)

Table properties > Show record count in Homepage

Check this option to display the count of accessible records of this table in the application homepage.

[back to top](#)

Table properties > Table group

Click this button to open the Table group dialog, where you can group tables for easier navigation.

[back to top](#)

Table properties > Table template

Click this button to open the Table view template dialog, where you can configure layout options of the table view for this table.

[back to top](#)

Table properties > Detail view title

The title of the detail view form. The detail view form is where users can edit and add data to the table.

[back to top](#)

Table properties > Redirect after insert

If you type a web address here, users will be sent to that address after they insert a new record. If you leave this box empty, users will be sent to the table view page. If you add the characters #ID# as part of the web address, they will get replaced by the primary key value of the newly inserted record. For example, these are valid addresses that you can put into the box: thanks.html

[back to top](#)

Table properties > Display a link to children records from

If you have lookup fields in other tables whose parent table is set to the current table, you'll find them listed here as children tables. If you check any of the children tables listed here, users will see a link to that table displayed in the detail view when a record is selected from the current (parent) table. This link displays records of the child table related to the current parent record. For example, you may have

an artists table, and a songs table. The songs table has an ‘Artist’ field whose parent table is the artists table. Using this feature, users who select an artist from the artists table will see a ‘songs’ link in the detail view of the selected artist. Clicking on that songs link, users will see all the songs that belong to the selected artist.

[back to top](#)

Table properties > Default focus field

Specify which field (if any) is focused by default when the user opens the detail view.

[back to top](#)

Table properties > Enable detail view

Use this option to control whether users can see the detail view or not.

[back to top](#)

Table properties > Allow detail print-view

If you check this option, users will see a ‘Print preview’ button in the detail view when selecting a record. Clicking that button would display a printer-friendly view of the selected record.

[back to top](#)

Table properties > Hide ‘Save As Copy’ when editing

By default, when a user selects a record for editing, the ‘Save As Copy’ button is displayed in the detail view so that the user can save a copy of the selected field. Checking this option would disable this behavior and the ‘Save As Copy’ button won’t be displayed.

[back to top](#)

Table properties > Allow adding new records from Homepage

If you check this option, users with insert permission can directly add records to this table from the homepage, without having to navigate to the table and click the Add new button in there.

[back to top](#)

Table properties > Delete records even if they have children records

The default behavior when a user tries to delete a record is that the AppGini-generated code will check to see if this record has one or more child records (records that have lookup fields pointing to the record to be deleted). If one or more children are found, the record is not deleted.

[back to top](#)

Table properties > Display detail view in a separate page

If you check this option, the detail view (the form where users can add or insert records) will be displayed in a separate page instead of below the table view.

[back to top](#)

Table properties > Keep action buttons visible while scrolling down

For long detail view forms, we recommend checking this option so that the action buttons (Save changes, Back, Print preview, Delete, .. etc) are always visible to the user, without having to scroll up/down to find them.

[back to top](#)

Table properties > Table technical documentation

This section is where you can edit and preview the technical documentation of this table. The technical documentation is useful for adding notes and technical descriptions for your project, tables and fields. It's not visible to your app users, but can be viewable in the admin area if the option Show technical documentation page in the admin area under project properties is checked.

[back to top](#)

Table properties > Edit Technical Documentation

Click this button to open the technical documentation editor for this table.

[back to top](#)

Table properties > Technical documentation preview

Here you can see a preview of the technical documentation for this table.

[back to top](#)

Table properties > Parent/Children settings

This feature is enabled only if the current table has children tables. Displays settings for showing the children records in the detail view.

[back to top](#)

Table properties > Table icon

Click this icon to select a new icon for the table. The table icon is displayed in the homepage and the navigation menu of the generated application.

[back to top](#)

Search box

You can use this box to search for a specific table/field by typing its name or part of it then clicking the lens icon. Click it again to move to the next matching table/field.

[back to top](#)

File menu

Includes commands for starting a new project, opening an existing one, importing from an existing MySQL database or a CSV file, and saving current project.

[back to top](#)

Tables menu

Includes commands for adding and deleting tables and fields from the current project.

[back to top](#)

Project menu

Includes commands for generating the web application, and for changing the application theme.

[back to top](#)

Add-ons menu

Open this menu to see a list of available AppGini add-ons and plugins that can enhance your project and add more functions to it.

[back to top](#)

Help menu

Includes various help resources

[back to top](#)

Copy

Copies the selected table or field to the clipboard

[back to top](#)

Paste

Pastes a copy of the field or table in the clipboard to the project

[back to top](#)

Toggle Highlight

Marks/unmarks the current table or field with a yellow background for easily returning to it later on.

[back to top](#)

Activate/deactivate help for toolbar icons

Click this icon to switch to help mode where clicking any icon would show its help rather than execute its action. Click again to switch to normal mode.

[back to top](#)

Working with table fields

How can I add a field to a project?

Each table in your AppGini project would include at least one field. To create a new field, select a table from the Project Browser and then click the ‘New Field’ tool bar icon or open the Tables menu > Fields > New Field. You might also use CTRL + F keyboard shortcut.

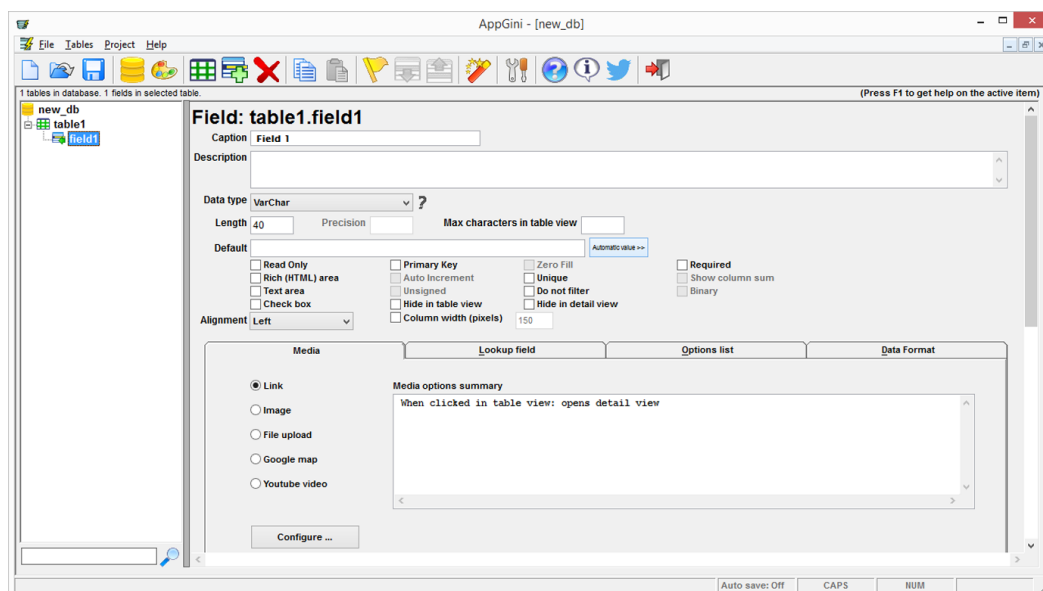


Figure 14: New Field

How can I rename a field?

Select the field by clicking on it in the Project Browser (if it is not already highlighted) and click on its name or press F2. A cursor will appear allowing you to rename the field.

Tip: AppGini handles some field names in a special manner. For example, if you name your field “id”, AppGini will automatically set it as an auto-increment primary key integer field. If you name it “comments”, AppGini will set it as a text field that is editable in a rich text box. Special names also include “date”, “*_date” (that is any name ending in “_date”), “description”, “photo”, “image” and “email”.

Can I change the order of fields in a table?

Yes. Click on a field in the left panel, and then click on the down or up arrow in the toolbar to move the field up or down. If you prefer to use the keyboard, select the field and press Ctrl+u (move the field up) or Ctrl+d (move the field down).

Can I clone/copy a field?

You can copy a field from another AppGini project or from the same project using the Copy and Paste icons from the top toolbar.

How can I delete a field?

Select the field by clicking on it in the Project Browser and press the Delete button.

What about field properties?

In AppGini, click with the mouse on any property and press F1 to view context help explaining the property.

Field properties pane

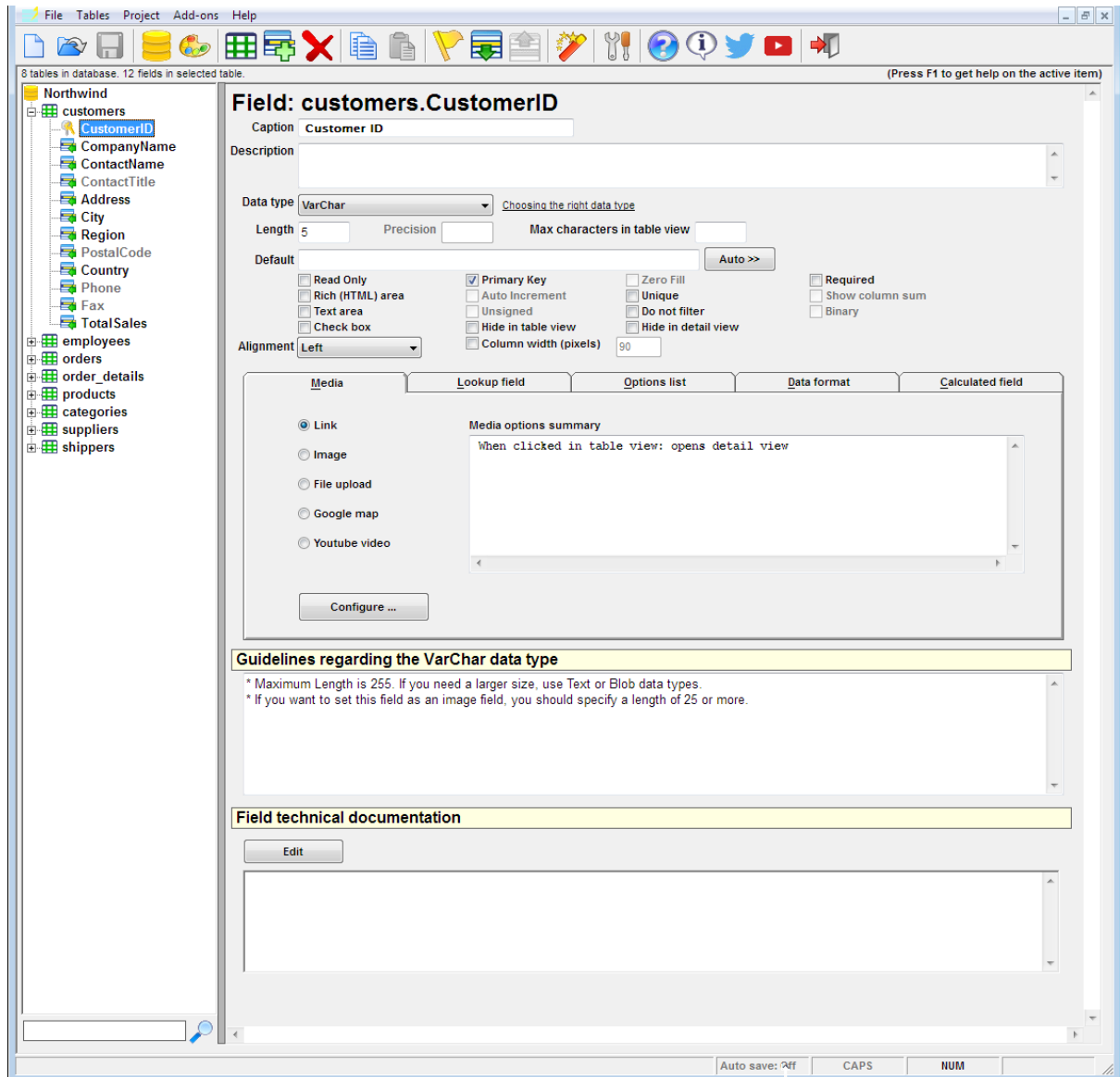


Figure 15: Field properties pane

Caption

The field caption is the title of the field as it would appear in the table view and the detail view of its table in the generated web application.

The Media Tab

This tab allows you to configure your field to be displayed as a web-link, an image, a file, a google map or even a YouTube video.

Link option

Configure the way your field behaves when clicked. It can be configured to open the detail view of the current record, a URL, an email link, or not be clickable at all.

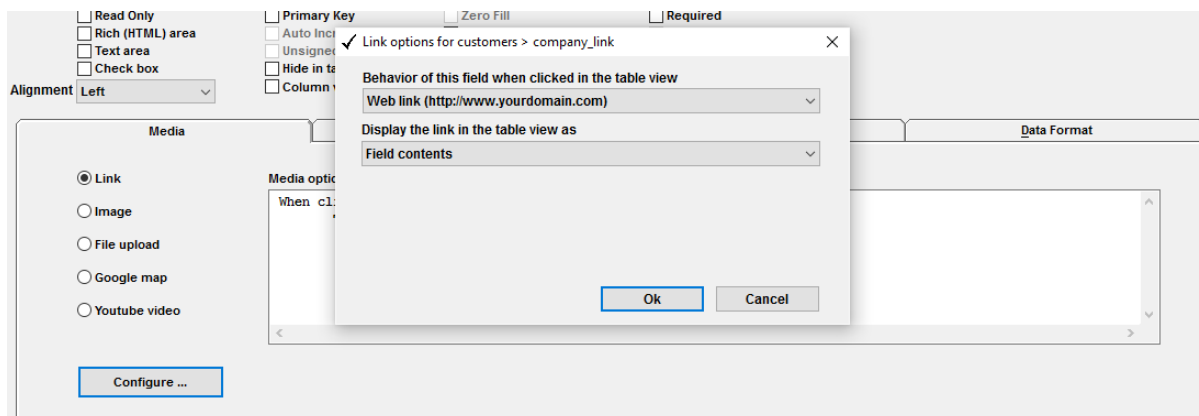


Figure 16: Link option

If you configure the field to display as a web-link and the user clicks that field, the link is opened in a new window.

The Image option

This option allows you to configure the field to be displayed as an image. You can allow users to upload jpg, jpeg, gif and png images. You can also configure the maximum allowed file size.

You can choose how to display the image. It can be displayed as a zoomable thumbnail image in the table and detail view. You can also configure the thumbnail size.

This is an example of how the image is displayed in the detail view.

And this is how the image is displayed in the table view.

The File upload option

This option allows the user to upload many different file types. You can configure the field to be displayed as the field content, clickable icon or contents of another page.

This is how the file upload field is displayed in the detail view.

You can configure the field to be displayed as the field content, clickable icon or contents of another field.

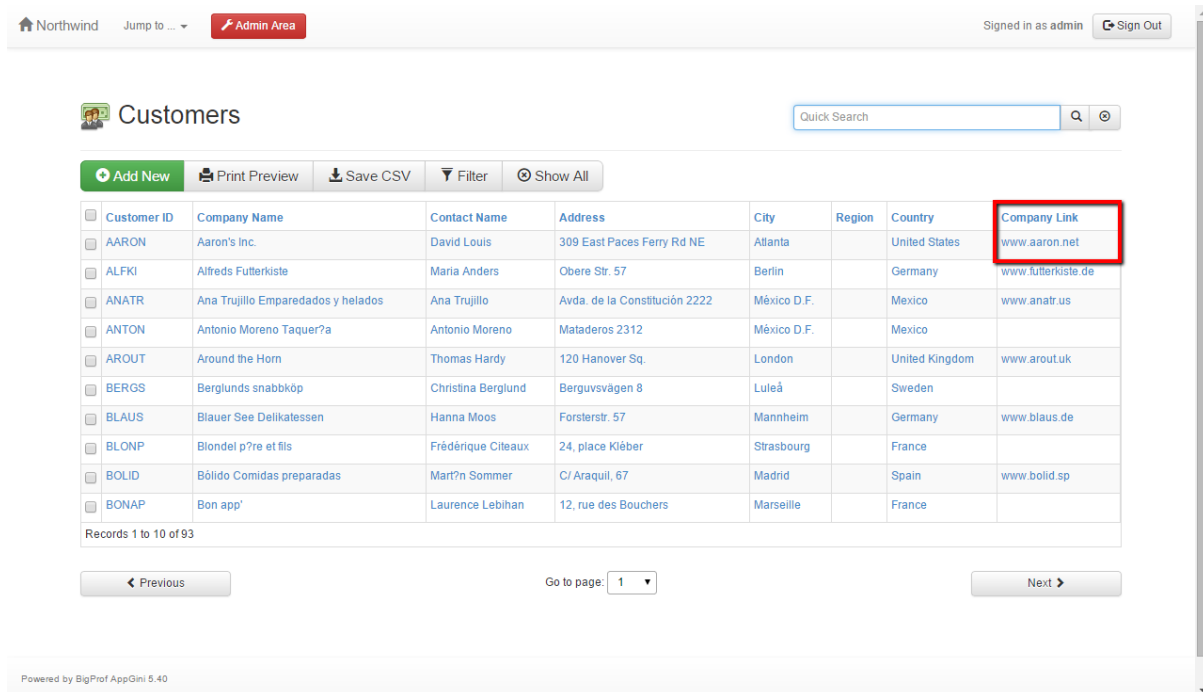


Figure 17: Link as displayed in the table view

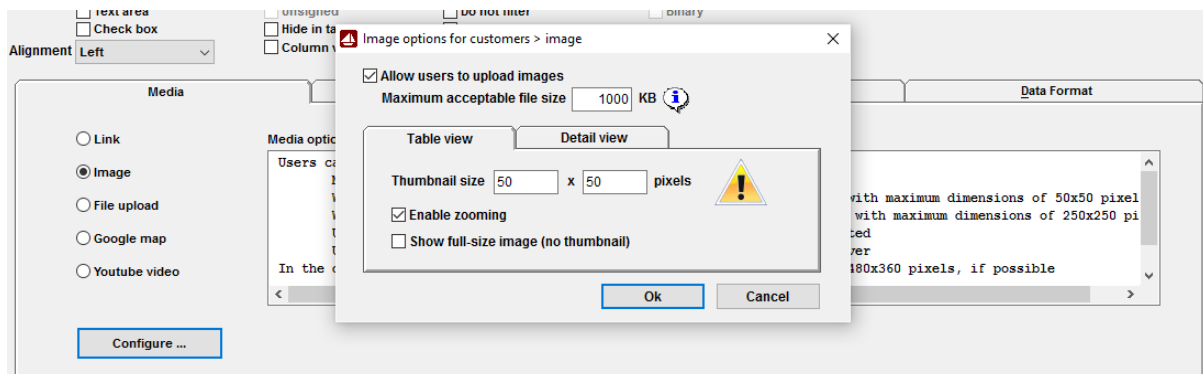


Figure 18: Image option



Figure 19: Image thumbnail in the detail view

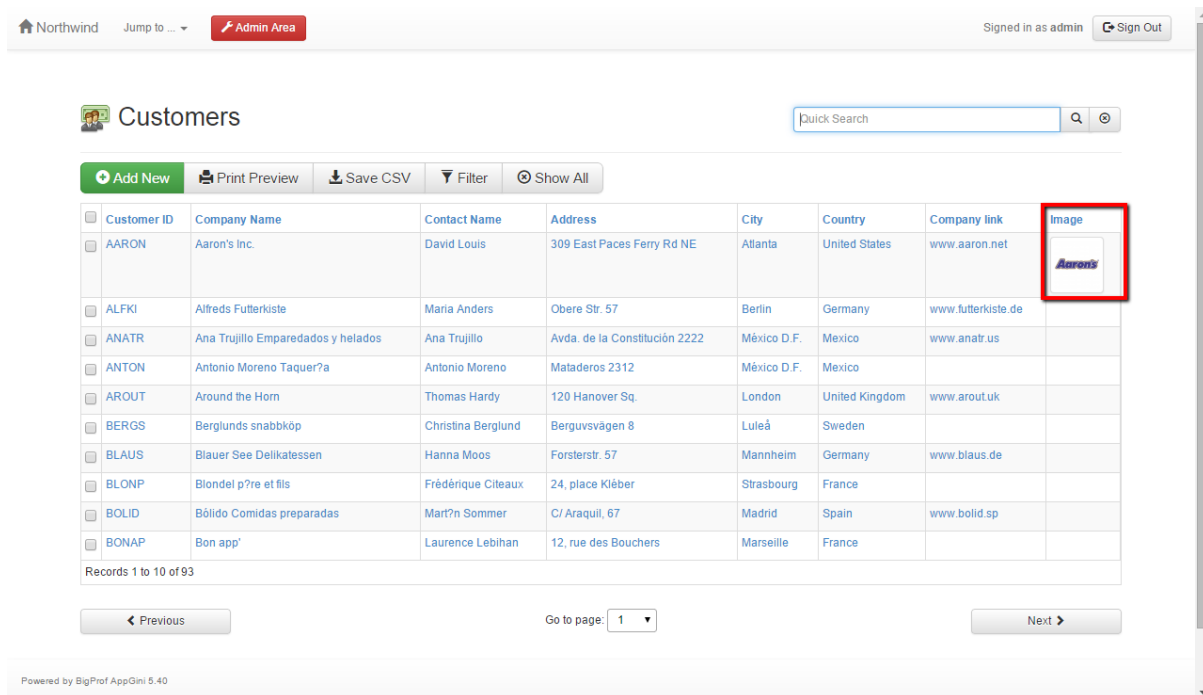


Figure 20: Image thumbnail in the table view

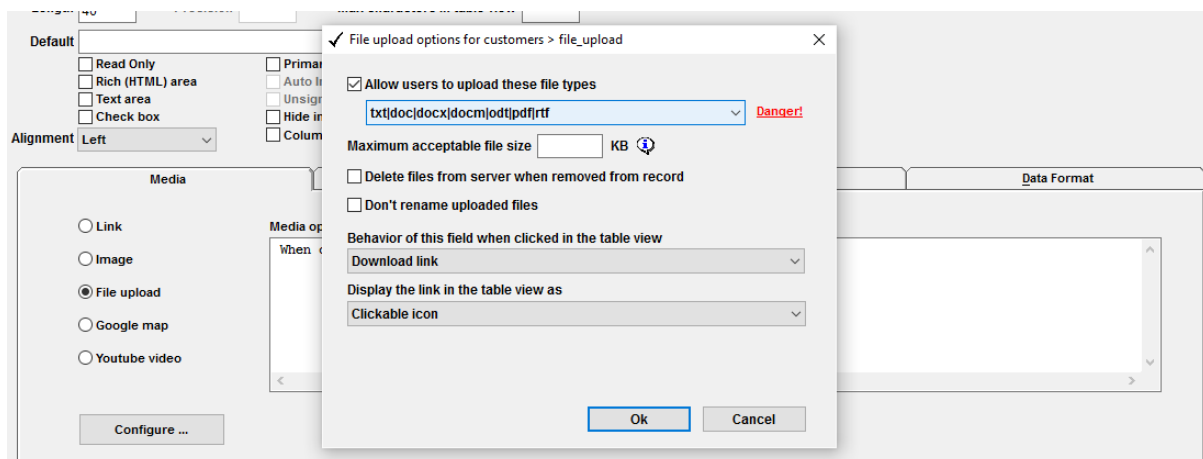


Figure 21: File upload option



Figure 22: File upload field in the detail view

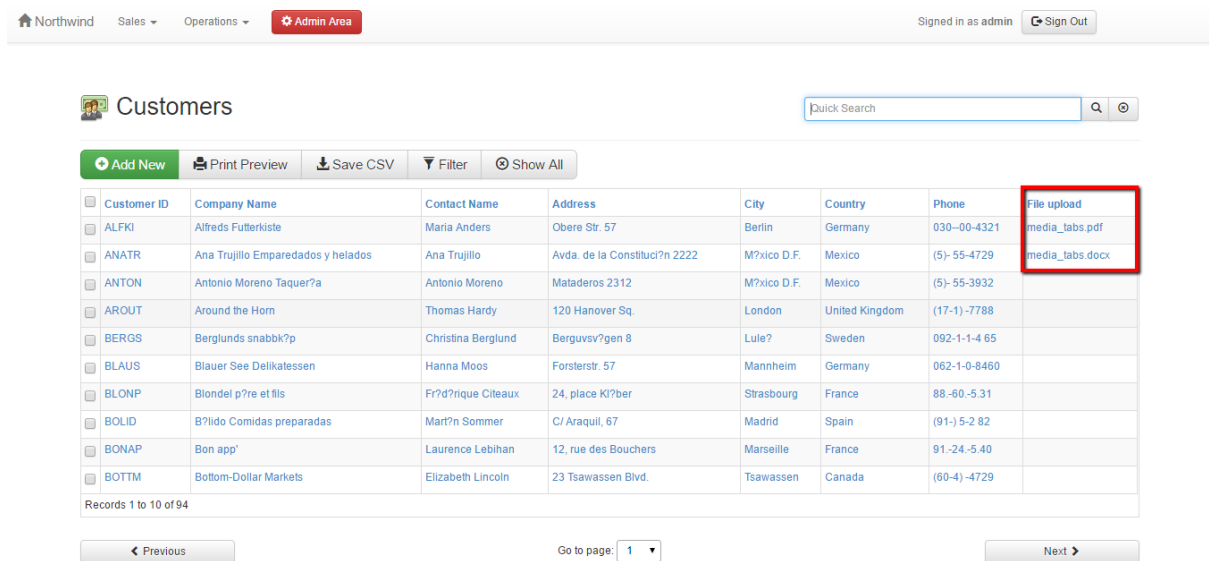


Figure 23: File upload field in the table view

Google Maps

You can add a google map to your records, simply by creating a new field having any textual data type and setting the field length to 200 or more.

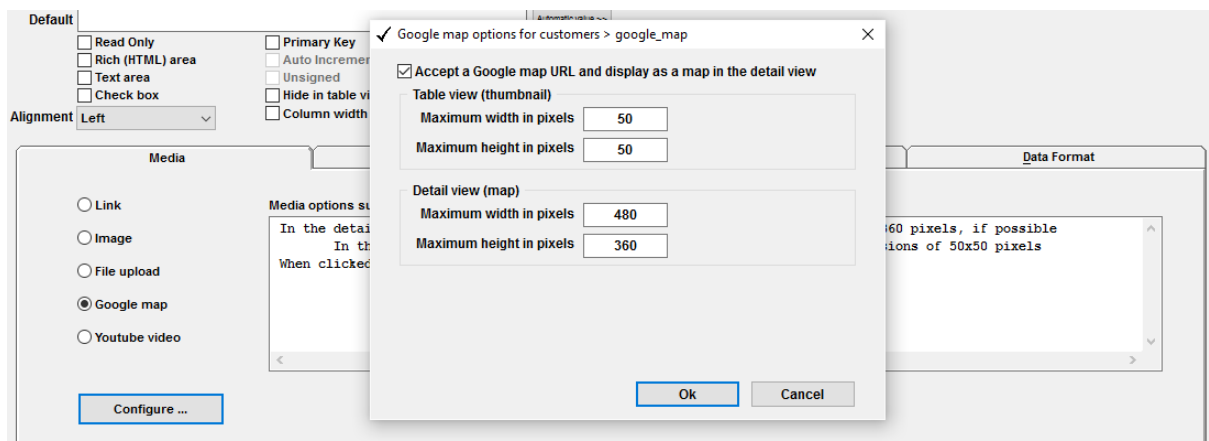


Figure 24: Google Map option

Google Maps require a Google API key to work correctly. You can add one by simply clicking the settings dialog to get a key. This allows you to insert interactive maps into your application.

You can configure how to display the Google map in the detail view as well as in the table view. Choose the size that meets your requirements.

In the table view, the map is displayed as a thumbnail image.

YouTube video

This field accepts a YouTube URL and displays it as a movie in the detail view.

You can configure how to display the YouTube video in the detail view and the table view. Here is an example of how the YouTube video is displayed in the detail view.

And this is how the YouTube video is displayed in the table view.

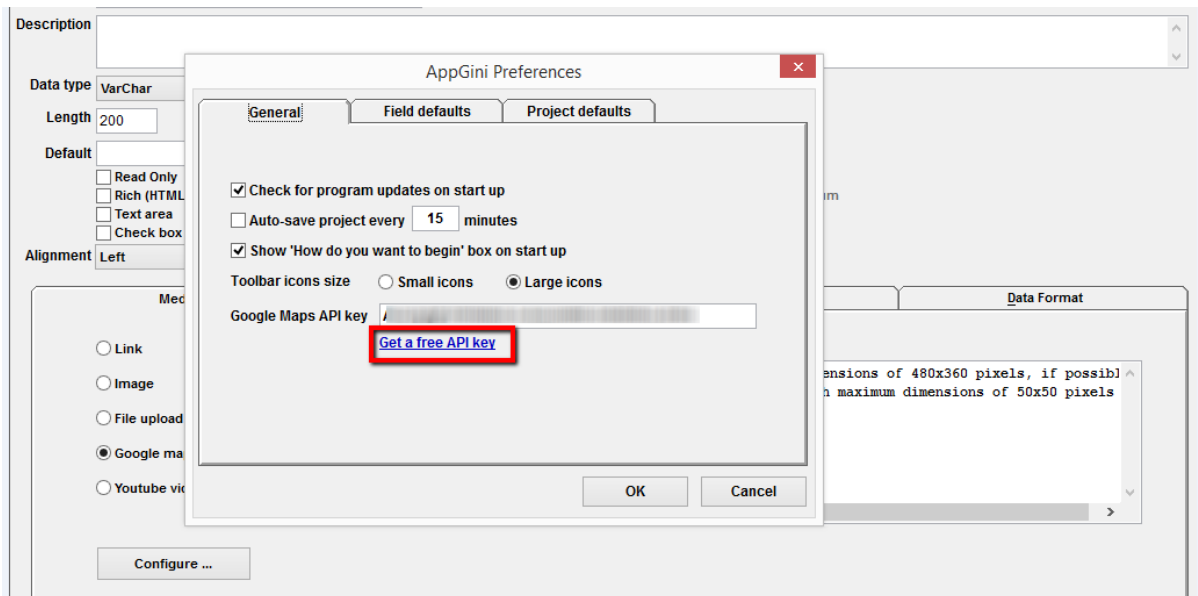


Figure 25: Google Maps API key in settings

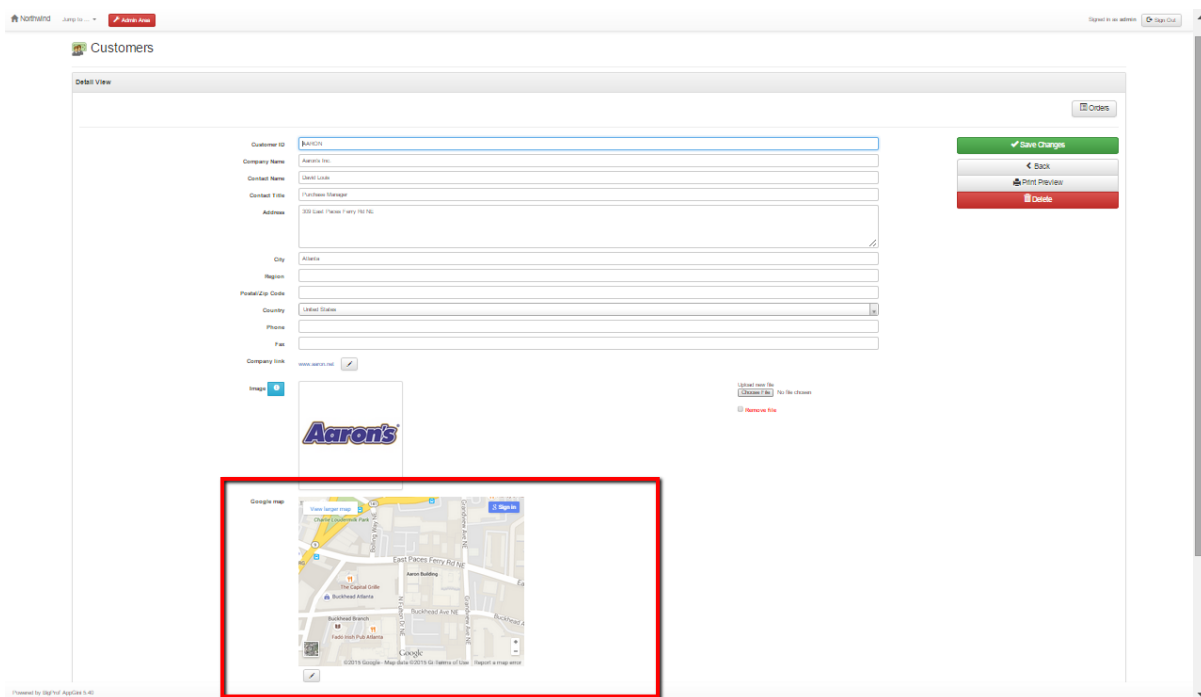




Figure 26: Google Map in the detail view

Northwind Jump to ... Admin Area Signed in as admin Sign Out

Customers

Quick Search

Add New Print Preview Save CSV Filter Show All

Customer ID	Company Name	Contact Name	Address	City	Region	Country	Company link	Image	Google map
AARON	Aaron's Inc.	David Louis	309 East Paces Ferry Rd NE	Atlanta		United States	www.aaron.net		
ALFKI	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin		Germany	www.futterkiste.de		
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.		Mexico	www.anatr.us		
ANTON	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.		Mexico			
AROUT	Around the Horn	Thomas Hardy	120 Hanover Sq.	London		United Kingdom	www.arout.uk		
BERGS	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå		Sweden			
BLAUS	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim		Germany	www.blaus.de		
BLONP	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg		France			
BOLID	Bólido Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid		Spain	www.bolid.sp		
BONAP	Bon app'	Laurence Leblan	12, rue des Bouchers	Marseille		France			

Records 1 to 10 of 93

Previous Go to page: 1 Next

Powered by BigProf AppGini 5.40

Figure 27: Google Map in the table view

Read Only Rich (HTML) area Text area Check box Alignment Left

Media

Link Image File upload Google map **YouTube video**

Configure ...

Primary Key Auto Increment Unsigned Hide in table Column width

Media options

In the detail view In the table view When clicked

Youtube options for customers > youtube_video

☒ Accept a Youtube URL and display as a movie in the detail view

Table view (thumbnail)

Maximum width in pixels 50

Maximum height in pixels 50

Detail view (video)

Maximum width in pixels 480

Maximum height in pixels 360

Ok Cancel


Data Format


1080x360 pixels, if possible
divisions of 50x50 pixels

Figure 28: YouTube video option

Country

Phone

File upload  Upload new file
 No file chosen

Google map 

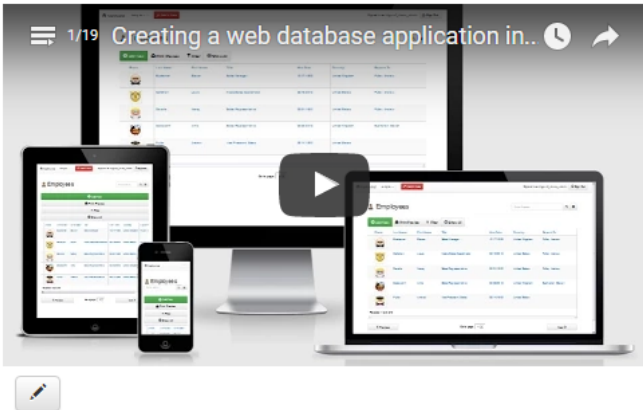
Youtube video 




Figure 29: YouTube video in the detail view

Northwind Jump to ... Admin Area Signed in as admin Sign Out

Customers

Quick Search

[Add New](#)
[Print Preview](#)
[Save CSV](#)
[Filter](#)
[Show All](#)

<input type="checkbox"/>	Customer ID	Company Name	Contact Name	Address	City	Country	Company link	Image	Google map	Youtube video
<input type="checkbox"/>	AARON	Aaron's Inc.	David Louis	309 East Paces Ferry Rd NE	Atlanta	United States	www.aaron.net			
<input type="checkbox"/>	ALFKI	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	Germany	www.futterkiste.de			
<input type="checkbox"/>	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	Mexico	www.anatr.us			
<input type="checkbox"/>	ANTON	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	Mexico				
<input type="checkbox"/>	AROUT	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	United Kingdom	www.arout.uk			
<input type="checkbox"/>	BERGS	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	Sweden				
<input type="checkbox"/>	BLAUS	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	Germany	www.blaus.de			
<input type="checkbox"/>	BLOMP	Blondel p?re et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	France				
<input type="checkbox"/>	BOLID	Bólido Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid	Spain	www.bolid.sp			
<input type="checkbox"/>	BONAP	Bon app'	Laurence Leblanc	12, rue des Bouchers	Marseille	France				

Records 1 to 10 of 93

[Previous](#)
Go to page:
[Next](#)

Powered by BigProf AppGini 5.40

Figure 30: YouTube video in the table view

Understanding lookup fields

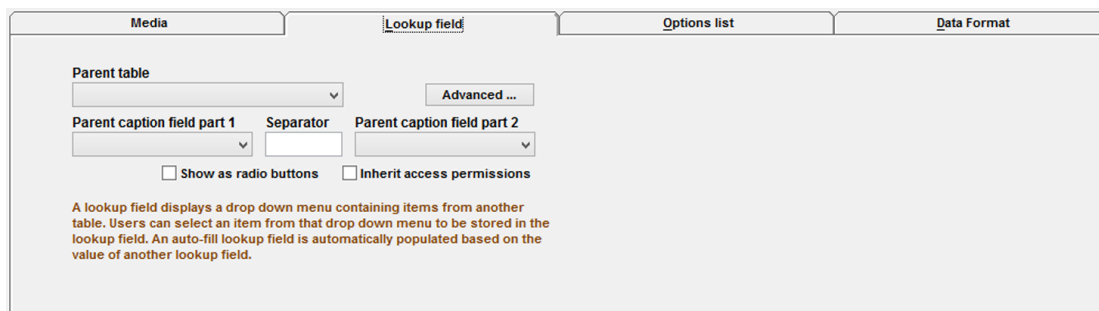


Figure 31: Lookup field tab in AppGini

A lookup field (also known as a foreign key) is how AppGini links 2 fields from 2 tables together. For example, let's say that our database contains a products table, a suppliers table, and a product categories table.

Tip: The list of video tours to the left of this page contains some helpful videos explaining several features of lookup fields. It takes less than 20 minutes to watch them all. So, please do.

The products table stores data about each product, including the supplier of the product, and the product category. Since suppliers and categories are stored in their own tables, the products table should look up those two tables when storing supplier and category data for each product.

The products table is thus a child table that has 2 parent tables: suppliers and categories. To achieve this, we should create a field in the products table to hold supplier data, and another one to hold category data. Each of these two fields is called a lookup field. We can define its properties in the Lookup field tab of the field properties pane, which is shown above. Lookup fields are also known as foreign key fields.

How will a lookup field appear in the generated application?

The above screenshot shows the detail view of the products table as generated by AppGini. The detail view is where users can edit records of the table. The “Supplier” and “Category” fields are lookup fields that bring their data from the suppliers and products tables, respectively. This data is represented in a drop down menu for each field.

How to set up a lookup field?

To set a field as a lookup field in AppGini, create a new field and, in its properties pane, go to the “Lookup field” tab, as displayed in the above screenshot. From the “Parent table” drop down, select the table that contains the source data. From the “Parent caption field part 1” drop down, select the source field.

You can optionally specify a second source field to be joined to the first one. For example, you could create a lookup field that lists the full name by joining a “first name” field to a “last name” field, using a space as the separator.

Note: AppGini will change the data type of the lookup field to be the same as that of the primary key of the parent table. This is normal behavior and you shouldn't alter it. If the parent table doesn't have a

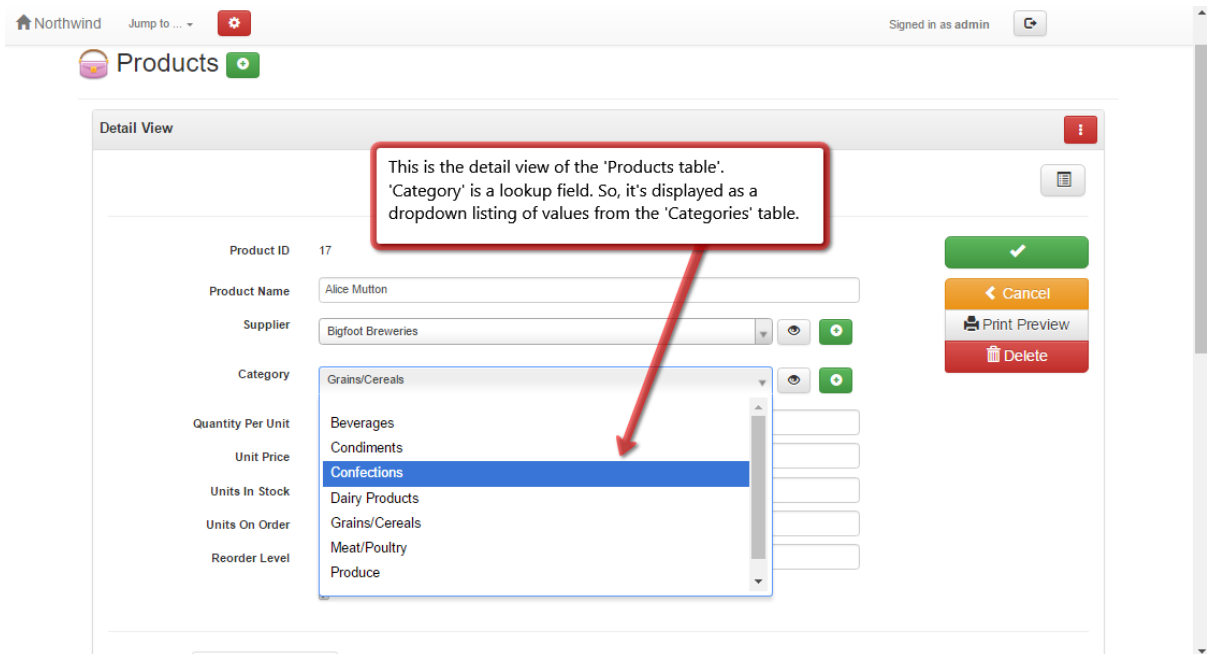


Figure 32: Lookup field in the detail view of the generated application

Media	Lookup field	Options list	Data Format
<p>Parent table</p> <p>categories</p> <p>Advanced ...</p> <p>Parent caption field part 1 Separator Parent caption field part 2</p> <p>Description</p> <p><input type="checkbox"/> Show as radio buttons <input type="checkbox"/> Inherit access permissions</p> <p>A lookup field displays a drop down menu containing items from another table. Users can select an item from that drop down menu to be stored in the lookup field. An auto-fill lookup field is automatically populated based on the value of another lookup field.</p>			

Figure 33: Lookup field properties in AppGini

primary key yet, you should change the data type of the lookup field manually to match the primary key once you create one.

Displaying lookup fields as an options list (radio buttons)



Figure 34: A lookup field displayed as radio buttons in the detail view

AppGini makes it possible to display the lookup field as an options list (radio buttons list) rather than a drop-down menu, as shown above. To do so, simply check the “Show as radio buttons” option in AppGini, as shown below.

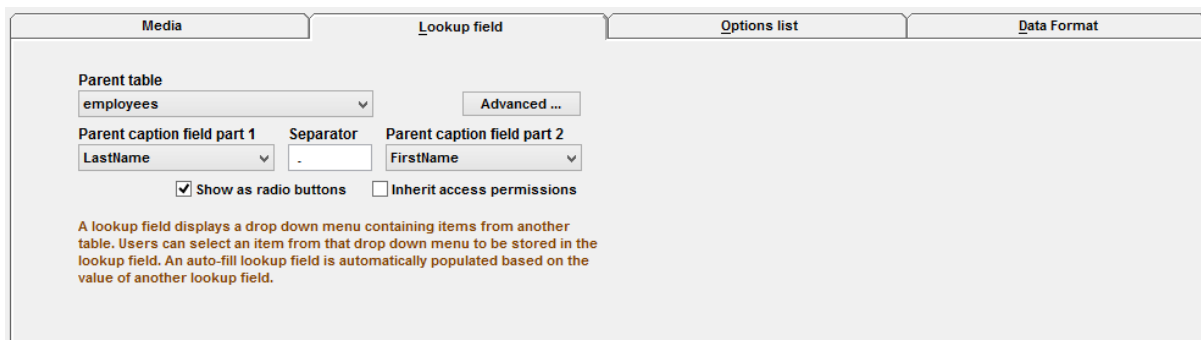


Figure 35: Displaying a lookup field as radio buttons in AppGini

Parent/Children settings

When you configure a field as a lookup field, the parent table you specify for that field can, in turn, be configured to show some special behavior. In AppGini, if you click the parent table, you should see a button labeled Parent/Children settings, like the one to the right.

Clicking that button displays the Parent/Children settings window - as shown below, which allows you to enable displaying child records below the detail view of the parent record.

This window lists all child tables of the current table (that is, tables that contains a lookup field where the parent table is set to the current table). Select a child table from the grid at the left to configure its related behavior in the parent table. An example of parent and child tables is the orders and order_details tables. The orders table is a parent table of order_details. Every order saved in the orders table would have one or more items saved in the order_details table.

Show tab below detail view would display a list of child records below the detail view when you select a parent record. For example, this is how an order looks like in the detail view, where the order items are listed at the bottom.

Copy child records when copying parent would copy child records if the user copies the parent record by clicking the Save As Copy button. The lookup field in the copied child records would be automatically set to the new parent record. This is a very handy feature for scenarios like duplicating an

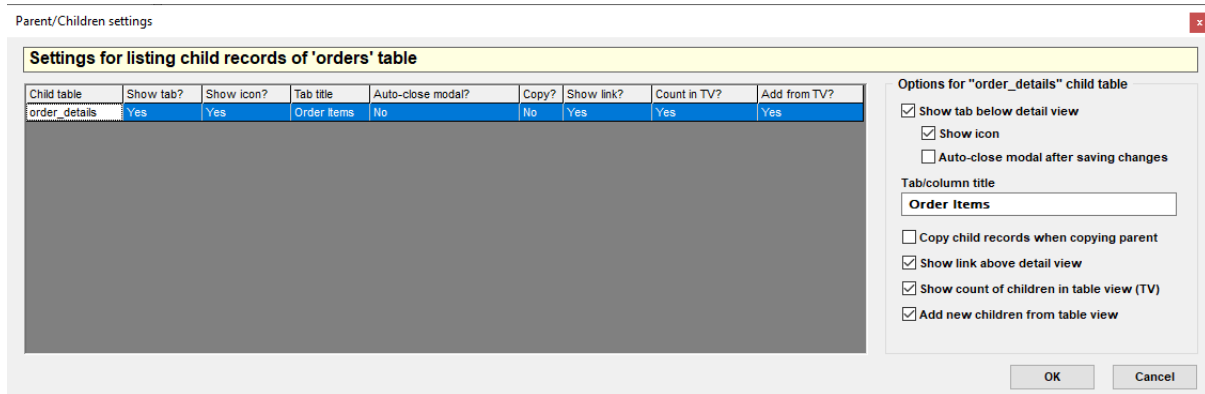


Figure 36: Parent/Children settings window in AppGini

order and all its order items, a product, and all the items in its bill of materials, ... etc. It saves users the time to manually add child records from scratch.

Copying child records requires cURL PHP extension to be installed and enabled on your server.

As of AppGini 5.81, a new configuration parameter, `$host`, was added in `config.php`. The value of this parameter is set by default to the host name of your server as automatically detected by PHP. However, if copying of child records is not working (and curl PHP extension is enabled), you might need to check and change this value manually by editing the config file. PHP might not detect the internal host name correctly in cases where servers are behind NAT or load balancers, Docker containers, or similar network configurations.

Display child info in the table view. Starting with AppGini 23.15, the Parent/Children settings dialog includes the option **Show count of children in table view**. Enabling this option displays the count of child records in the table view. You can also add new child records directly from the table view by enabling the option **Add new children from table view**. The screenshot below shows how both options would be displayed in the table view of the orders table, showing the count of order items in each order, and allowing users to add new items directly from the table view of orders.

See also the related video tutorial

Related screencasts

AppGini lookup fields and master detail pages


Using auto-fill look-up fields to automatically populate fields from another table

Creating cascading drop downs with AppGini

Displaying child info (count + add new) in the table view


Your browser does not support the video tag.

Detail View
⋮


 Order Items

Order ID
236188


Customer


+


Employee


+


Order Date




Required Date



Shipped Date



Ship Via


+

Freight


✓ Save Changes


← Back











🖨️ Print Preview

🗑️ Delete

➦ Save As Copy


Order Items


Add New


	Order ID	Product	Category	Unit Price	Quantity	Discount
	236188	Côte de Blaye	Beverages / Aux Joyeux ecclésiastiques	\$263.50	26	\$0.00
	236188	Filo Mix	Grains/Cereals / G'day, Mate	\$7.00	16	\$0.00
	236188	Genen Shoyu	Condiments / Mayumi's	\$15.50	19	\$0.00
	236188	Sir Rodney's Scones	Confections / Specialty Biscuits, Ltd.	\$10.00	6	\$0.00
	236188	Northwoods Cranberry Sauce	Condiments / Grandma Kelly's Homestead	\$40.00	1	\$0.00
	236188	Chocolade	Confections / Zaanse Snoepfabriek	\$12.75	3	\$0.00
	236188	Pavlova	Confections / Pavlova, Ltd.	\$17.45	19	\$0.00
	236188	Inlagd Sill	Seafood / Svensk Sjöföda AB	\$19.00	18	\$0.00
	236188	Longlife Tofu	Produce / Tokyo Traders	\$10.00	17	\$0.00
	236188	Original Frankfurter grüne Soße	Condiments / Plutzer Lebensmittelgroßmärkte AG	\$13.00	15	\$0.00

Records 1 to 10 of 10

←
→

Figure 37: Child records displayed below the detail view of a parent record

 **Orders** Quick Search

[Add New](#)
[Print Preview](#)
[Save CSV](#)
[Filter](#)

<input type="checkbox"/>	Order ID	Customer					Order Items
<input type="checkbox"/>	11077	Rattlesnake Canyon Grocery				1373.91	25 +
<input type="checkbox"/>	11076	Bon app'	04/14/2023			1056.25	3 +
<input type="checkbox"/>	11075	Richter Supermarket	04/14/2023			586.00	3 +
<input type="checkbox"/>	11074	Simons bistro	04/14/2023			244.25	1 +
<input type="checkbox"/>	11073	Pericles Comidas clásicas	04/13/2023			300.00	2 +
<input type="checkbox"/>	11072	Ernst Handel	04/13/2023			5218.00	4 +
<input type="checkbox"/>	11071	LILA-Supermercado	04/13/2023			509.90	2 +
<input type="checkbox"/>	11070	Lehmanns Marktstand	04/13/2023			1873.05	4 +
<input type="checkbox"/>	11069	Tortuga Restaurante	04/12/2023	04/14/2023		360.00	1 +
<input type="checkbox"/>	11068	Queen Cozinha	04/12/2023			2384.35	3 +

Records 1 to 10 of 830

Figure 38: Displaying child info in the table view

Calculated fields

Calculated fields feature is available in AppGini 5.80 and above.

What are calculated fields?

As of AppGini 5.80, you can now configure one or more fields in your app as *calculated fields*. Calculated fields are read-only fields that get populated automatically with a value calculated from any formula you specify. The formula for a calculated field must be a MySQL-compatible SQL query that returns a single value. The value returned from the SQL query is saved to the calculated field whenever the record containing that field is accessed by users.

Calculated fields can be very helpful in numerous scenarios. For example, to automatically calculate and update the subtotal and total of an invoice, number of students enrolled to a course, average score of course, due date of an invoice (for example if you want to set a business rule to set a due date of an invoice to 15 days after issue date), most recent status of a shipment, flag overdue tasks, indicate if prospect customer should be contacted today ... etc. There are endless possibilities to applying calculated fields.

Conditions for a field to become a calculated field

If you are trying to set a field as a calculated field, it must **NOT** meet any of the following conditions

- Fields not set as read only
- Primary key fields
- Required fields
- Text area and rich (HTML) area fields
- Auto-increment fields
- Unique fields
- Web/email link fields
- Image/file upload fields
- Map/video fields
- Lookup fields
- Options list fields
- Fields that have a data format specified (you can apply a data format in the calculation instead)
- Fields with default values (you can apply a default value in the calculation instead)

You'll see a clear error message in AppGini explaining why a field can't be set as a calculated field if any of the above conditions apply to that field.

Also, if you set a field as a calculated field, and later on make some changes to the field that prevent it from being a calculated field, you'll see a warning when generating the app that the calculation will be skipped, along with reason for skipping:

How to configure a calculated field

The basic steps are:

1. Create the field (if it already exists, make sure it meets the conditions above).
2. Set the field as read-only.

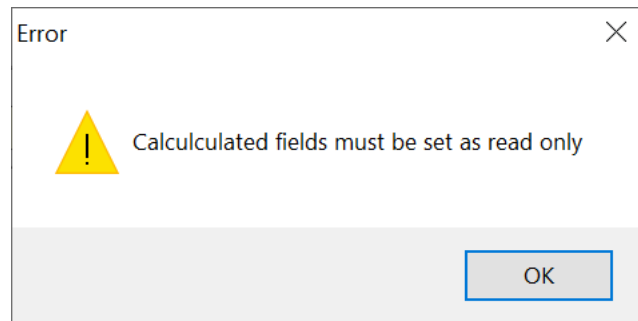


Figure 39: Calculated field error in AppGini.

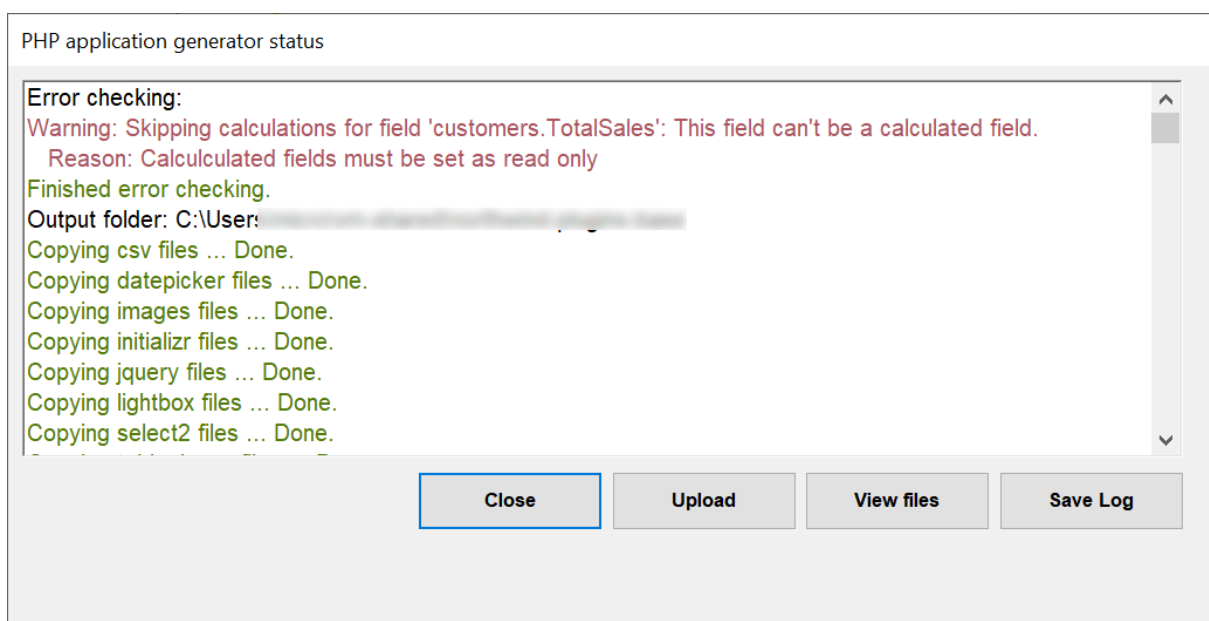
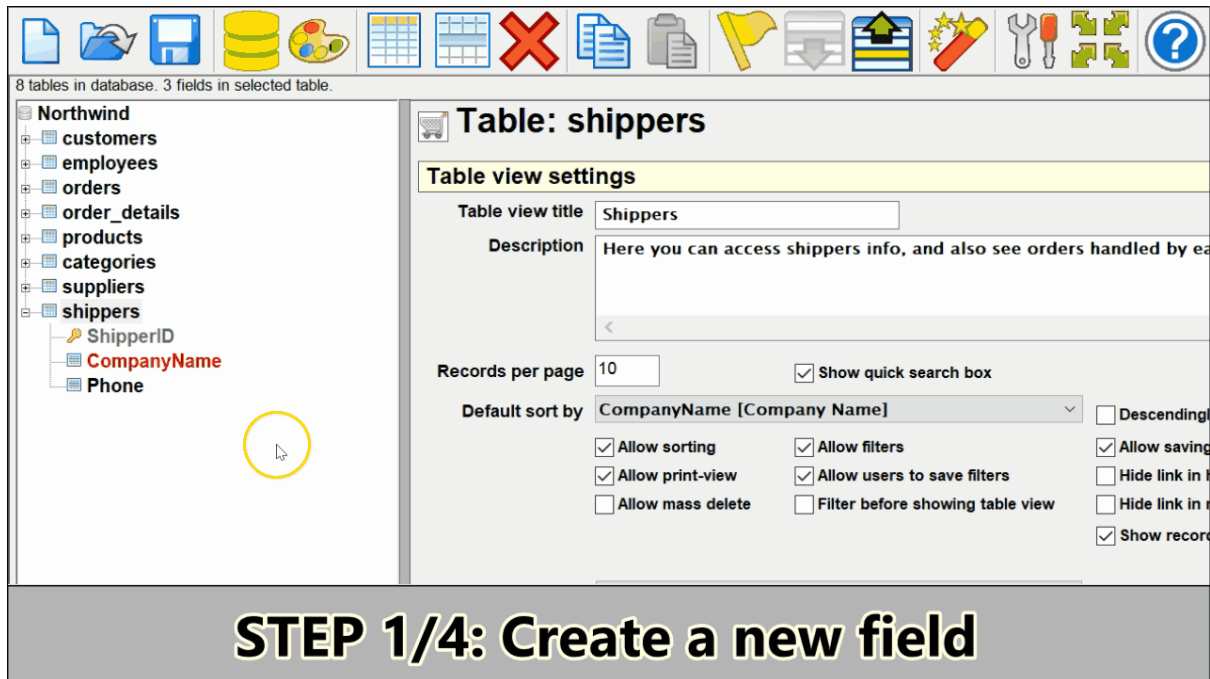


Figure 40: Calculated field skipped when generating an AppGini app

3. Navigate to the *Calculated field* tab and check the option *Automatically calculate the value of this field using the following SQL query*
4. Type the SQL query for calculating the field value.



Important note:

Valid SQL queries for calculated fields must be **SELECT** queries that return a single value. The returned value should be of the same data type as the calculated field. For example, this is a valid query to calculate the subtotal of an invoice line by multiplying the unit price by the quantity:

```
SELECT quantity * unit_price FROM invoice_items WHERE id='%ID%'
```

Special variables for use in calculated field queries

In the above query, we're using the special variable `%ID%`. When executing the query, this would be replaced by the primary key value of the current record. The following variables can be used in queries:

- `%ID%` Will be replaced with the ID (primary key) value of the current record before executing the query.
- `%USERNAME%` Will be replaced with the currently logged username before executing the query.
- `%GROUPID%` Will be replaced with the group ID of the currently logged username before executing the query.
- `%GROUP%` Will be replaced with the group name of the currently logged username before executing the query.
- `%TABLENAME%` Will be replaced with the name of the table containing the calculated field before executing the query.
- `%PKFIELD%` Will be replaced with the name of the primary key field of the table containing the calculated field before executing the query.

Please make sure to use single quotes around the above variables when using them in queries. You don't have to manually type the variable into the query in AppGini; you can place the cursor at the location where you want to insert the variable, and then click the desired variable at the right as shown in this screenshot:

The above special variables make it easy to write flexible queries that depend on the current user, group, or record. For example, you can use `%USERNAME%` to calculate the total sales made by the currently logged user, or use `%GROUP%` to calculate the total sales made by the group of the currently logged user. `%TABLENAME%` and `%PKFIELD%` can be used to write generic queries that can be copied and pasted to other tables without modification.

The query helper

For quicker and more precise query entry, we recommend using the *query helper*. Click the *Query helper* button below the query box to launch the query helper window, which looks like this:

The query helper window allows you to quickly insert various special code pieces into your SQL query. Just place the cursor at the position where you want to insert the piece of code, then choose the code you want to insert from the boxes at the right or the bottom of the query box, then click the *Insert* button.

You can insert special variables (as explained above), field names, SQL functions, or JOIN statements that join the table of the calculated field with one or more of its parent or child tables. This not only saves you time for manually typing these snippets, but also reduces typos and syntax errors.

Of course, using calculated fields requires some knowledge of SQL language, specifically SQL SELECT statement. There are many great SQL tutorials available online, as well as the official MySQL reference. We'll also list a few examples below that cover some widely-used scenarios. You can also ask for help from other users on our forum.

In the screencast below, we create a new 'Sales' field in the clients table, and configure it as a calculated field that displays the total of sales made to each client, by retrieving the sum of her paid invoices total.

We use the *Query helper* window to quickly and precisely write the query, including the join between the clients and invoices tables.

Debugging your query

You can easily debug your SQL query using phpMyAdmin or any similar MySQL admin utility. Select your database, then go to the SQL tab, where you can type or paste your SQL query. Replace %ID% with the primary key value of the record you wish to test. Also replace any other variables with their values, if needed. Then execute the query. The query should return a single value, and that should be the value you expect in your calculated field. If this is not the case, or if you see any error messages, you should edit the query and retry until no errors are shown and the expected value is returned.

Batch-updating calculated fields via command line

Added in AppGini 5.82

As described in the known issues below, calculated fields are normally updated only when users access them via the table or detail view. Sometimes, you want to update a large number of records without having to access each one. So, we added a command line script for doing that.

Command line means it can't be run from the browser. You can run it only from a terminal window, or install it to your server's crontab file to run it on a schedule.

If your app is hosted on your local Windows PC (for example using Xampp), you can open a command line terminal by opening the Windows Start menu and typing `cmd` then pressing Enter.

If your app is hosted remotely on a Linux server, you need to have shell access and connect to your server via an SSH client (for example PuTTY)

In both cases (Windows and Linux command line), you should navigate to the folder where your AppGini app is hosted and run this command:

```
php cli-update-calculated-fields.php
```

The above command would update all calculated fields in all tables. However, on large tables, this might take a long time. So, the command line tool provides several options to control its behavior as follows (you can get command line help on the tool by adding `-h` after the above command):

Supported arguments:

- t: comma-separated list of tables to update.
all tables will be updated if this argument is not specified
- s: comma-separated list of starting record numbers.
Default is 0 (beginning of each table)
- l: comma-separated list of records count to update in each table.
Default is records count - start
- x: comma-separated list of tables to exclude from updating,
overrides -t
- u: username to use in queries that have %USERNAME% placeholder.
Default is admin user
- h: displays this help message

Examples:

```
php cli-update-calculated-fields.php
```

Updates all records of all tables. Not recommended for large databases.

```
php cli-update-calculated-fields.php -s 2000 -l 1000
```

Updates 1000 records starting from rec# 2000 in all tables.

```
php cli-update-calculated-fields.php -t clients,orders -s 100,1000 -l 10,100
```

Updates records 100:110 of clients table and 1000:1100 of orders table.

```
php cli-update-calculated-fields.php -x clients
    Updates all records of all tables excluding clients table.
```

```
php cli-update-calculated-fields.php -u bob
    Updates all records of all tables as user bob.
```

Basic examples of calculated fields

For simple calculations performed on other fields of the same record, we'll list some common examples below.

Calculate subtotal for an invoice line by multiplying unit price and quantity

Let's assume you have an app for managing invoices. The invoice header (invoice number, due date, customer info, ... etc) is stored in the `invoices` table. Invoice lines (item, quantity, unit price, subtotal) are stored in the `invoice_items` table. You'd like to have the subtotal field automatically calculated when adding or editing an invoice line.

To set up this calculation, we'd check the read-only option for the `subtotal` field, then set it as a calculated field, and use this SQL query for calculating its value:

```
SELECT
`invoice_items`.`unit_price` * `invoice_items`.`quantity`
FROM `invoice_items`
WHERE `invoice_items`.`id` = '%ID%'
```

Automatic code by concatenating 2 or more fields

In some data entry scenarios, it's required to create an automatic code given one or more fields in the record. This is typical for product codes, inventory transaction codes, ... etc.

For example, let's assume we have a `products` table. When defining a new product, we'd like the product code to be the first 5 letters of the product `name` field, capitalized, followed by department ID, `dept_id`, followed by the characters SN, followed by the serial number as obtained from the auto-increment primary key field `product_id`. To set up this coding scheme, we'd create a read-only field, `product_code`, set its data type as `VarChar` of a suitable length, 50 or so, set it as a calculated field, and use this SQL query for calculating its value:

```
SELECT CONCAT(
    UPPER(SUBSTRING(`products`.`name`, 1, 5)),
    `products`.`dept_id`,
    'SN',
    `products`.`product_id`
) FROM `products`
WHERE `products`.`product_id` = '%ID%'
```

When a user defines a new product and saves it, the `product_code` field would be set automatically based on the above query to something like CELLP22SN302.

More advanced examples of calculated fields

Updating batch status to 'Consumable', 'Warning' or 'Expired' based on expiry date

In this example, let's assume we have an inventory app, and for each batch of items we add to inventory, we have an expiry date. We want to automatically update the `status` field of each batch to `Consumable` if expiry date is 30 days or more ahead, `Warning` if less than 30 days ahead, or `Expired` if expiry date is today or older.

To do so, we should configure the `status` field as read-only, calculated field, and use a query like this for the calculation:

```
SELECT IF(
  DATEDIFF(`expiry_date`, NOW()) > 30,
  'Consumable',
  IF(
    DATEDIFF(`expiry_date`, NOW()) > 0,
    'Warning',
    'Expired'
  )
) FROM `batches`
WHERE `id` = '%ID%'
```

Here is a brief explanation of the above query: The SQL `IF()` function accepts 3 parameters: a condition to check, and a value to return if the condition is true, and a value to return if false. For example, `IF(10 > 1, 'yes', 'no')` checks if 10 is greater than 1, and returns either 'yes' if true or 'no' if false. Of course, this should return 'yes'. In the above query, we *nested* 2 `IF` expressions to evaluate 3 cases rather than just 2. `DATE_DIFF()` accepts 2 dates and returns the difference between them in days. `NOW()` returns the current date/time.

Invoice subtotal by summing subtotals of invoice items

In this example, let's assume we have an `invoices` table that includes a `subtotal` field. We want to calculate the invoice subtotal by summing the subtotals of all lines of the invoice (each line contains a unit price and a quantity that we want to multiply to obtain the line subtotal). We should set the `subtotal` field as read-only calculated field. And here is a query we can use to perform this calculation:

```
SELECT
  SUM(`invoice_items`.`unit_price` * `invoice_items`.`quantity`)
FROM
  `invoices` LEFT JOIN
  `invoice_items` ON `invoices`.`id` = `invoice_items`.`invoice_id`
WHERE
  `invoices`.`id` = '%ID%'
```

In the above query, we are *joining* the `invoices` table and the `invoice_items` table. Those 2 tables are linked through the `invoice_id` lookup field in `invoice_items` table. Line 5 above performs this join. Line 2 multiplies each invoice item's quantity by its unit price and returns the sum for all items in the current invoice.

Looking for more help with queries?

Calculated fields is a very powerful tool, and there are so many different usage scenarios. We tried to cover some common use cases above, but if you need more help, feel free to post your usage case on our forum. We'll be frequently updating this page with more usage cases, so try also checking it later.

Please see this forum topic for some excellent notes on calculated fields by Jan from bizzworxx (thanks Jan!).

Known issues

The following limitations apply to calculated fields:

- Calculated fields are re-evaluated every time the record or its child records are accessed in the table view, the detail view, the print preview or the child table view. This could cause some performance issues for complex queries. This can be resolved using MySQL query caching.
- Calculated fields are evaluated only when their records/child records are accessed. If data that affects the calculation is changed, and you then retrieve the value stored in the calculated field

through a third-party app, it won't reflect the changes until it's accessed through your AppGini app itself.

- Similarly, if the calculated field is used as a parent caption field for a lookup field in another table, the lookup drop-down might not display the most up-to-date calculated values until the records of the calculated field are accessed in your AppGini app.

The simple work-around for the second and third issues above is to access the record(s) containing the calculated field in the table view in your AppGini app to update them.

Working with styles

AppGini offers you the flexibility to control the look of the generated application using CSS (Cascading Style Sheets). Click on the **Application theme** icon on the tool bar or from the Project menu select **Themes** . The window below will appear.

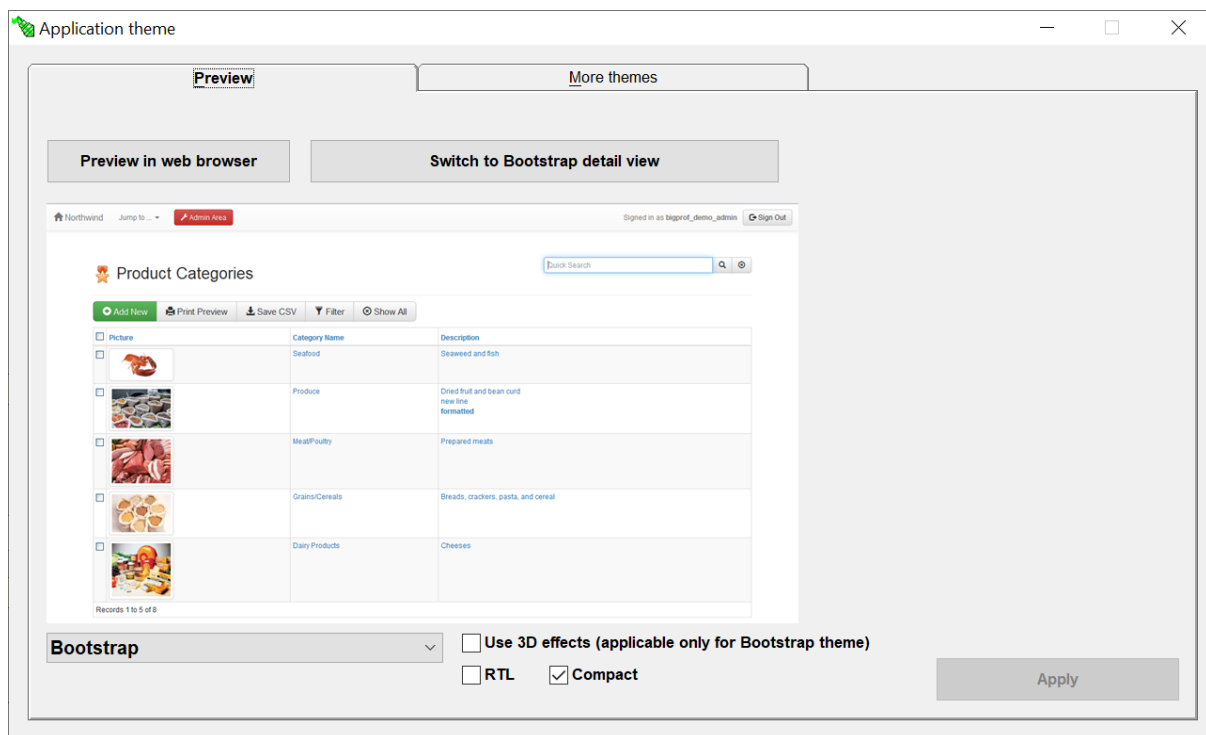


Figure 41: Application themes window

You can select a theme from the drop-down menu at the bottom.

Preview in web browser

You can click on this button to preview the selected theme in your web browser. This will display a page from the Northwind demo, with the selected theme applied to it.

Generating the PHP application



Figure 42: The ‘Generate App’ icon

After you have finished working with your project (defining tables, fields and styles) the only thing remaining now is firing your application. Click the ‘Generate App’ icon. You’ll be asked to select (or create) an output folder.

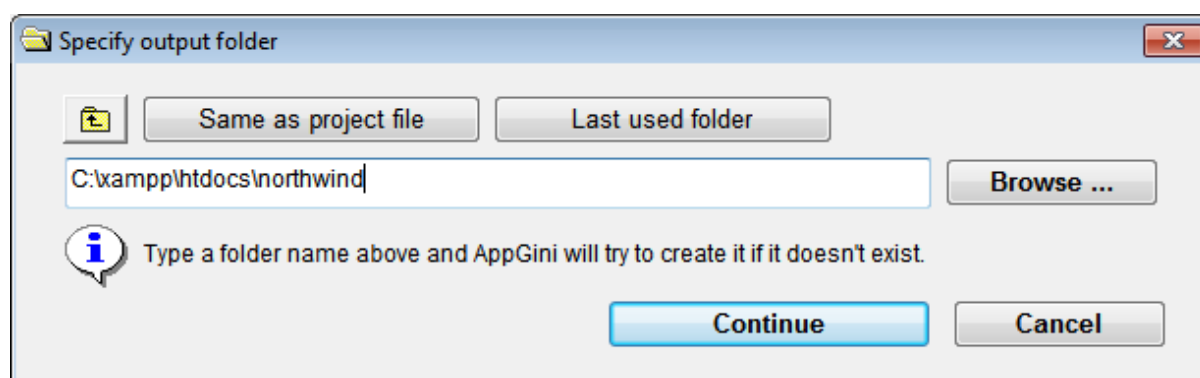


Figure 43: ‘Specify output folder’ dialog

If you’ve generated code for the open project before, this dialog will, by default, display the same output folder used before. You can select the output folder using the *Browse ...* button to navigate to any other folder.

By clicking the yellow folder button, you can change the output folder to the parent folder of the one currently specified. *Same as project file* would set the output folder to the same as the folder containing the AXP project file, and *Last used folder* would set the output folder to the one you used last (beware! this might contain an app generated by a different project file).

If you choose a folder that already contains previously-generated code, you’ll see a window that lists all the files that will be generated. You can specify in this window (shown below) which files to overwrite and which to skip.

Finally, a log window (shown below) reports events that happened during file generation: error checking, files overwritten, files skipped, failed files, and instructions for deploying the generated application. You can save the log for future reference if you click the “Save log” button. At this point you are finished with AppGini. The next step is to upload and set up your PHP application.

Tip: If you want to customize some of the generated files and don’t want AppGini to overwrite them if you regenerate your project later, set them as read-only. This is a very easy way of retaining your customized code. AppGini will just report that it couldn’t overwrite that file, and will continue generating the other files normally.

For more advanced code management, you should consider using hooks. Hooks allow you to add more

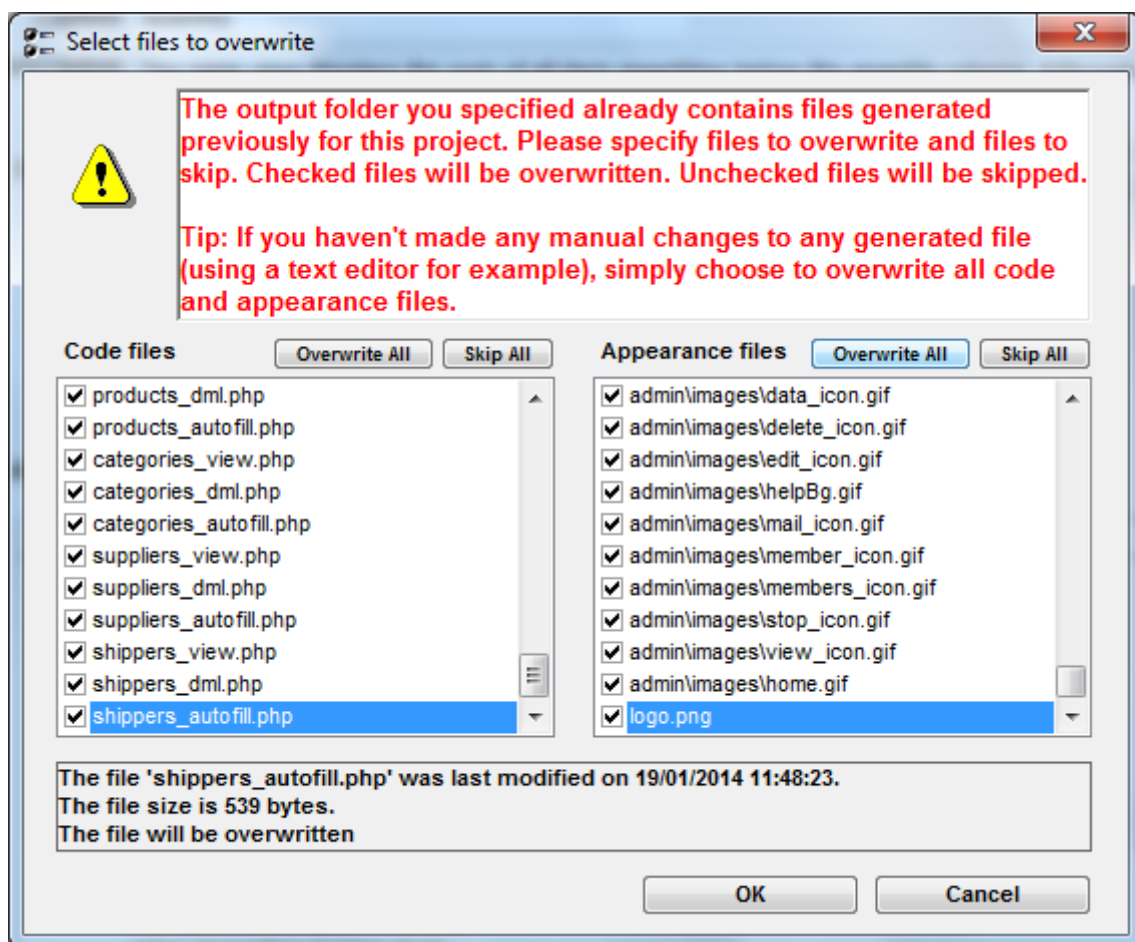


Figure 44: 'Select files to overwrite' dialog

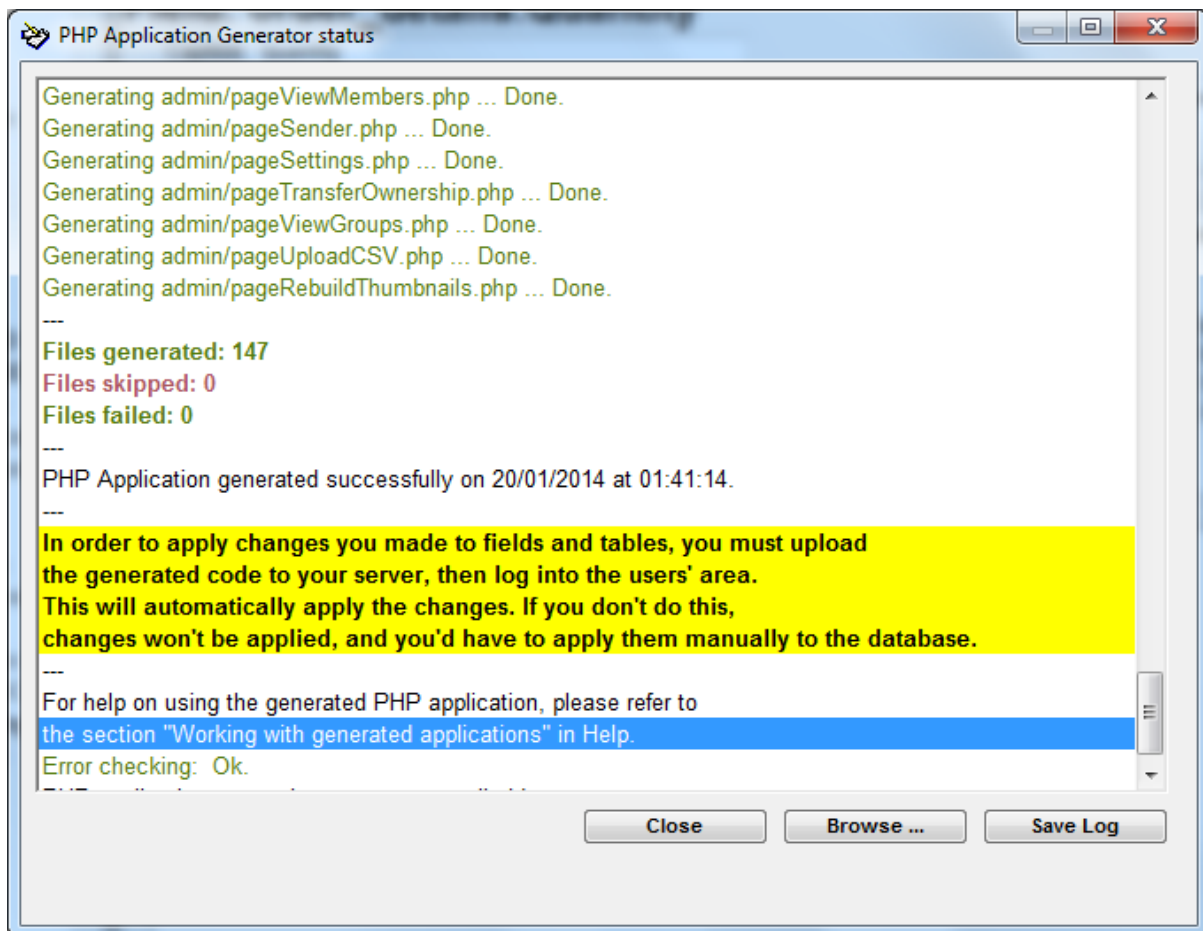


Figure 45: Log of generated files

functionality and customize your application behavior without losing your customizations whenever you regenerate the application later.

Keyboard shortcuts in AppGini

There are several keyboard shortcuts in AppGini that will help you work even faster with projects. Here is a list of them.

- **Ctrl + N**: Create a new project.
- **Ctrl + O**: Open an existing project.
- **Ctrl + S**: Save the current project.
- **Ctrl + Q**: Quit AppGini.
- **Ctrl + T**: Create a new table.
- **Ctrl + F**: Create a new field.
- **F2**: Rename the selected table or field.
- **F5**: Generate the application.
 - > Tip: Hold **Ctrl** while clicking the “Generate AppGini app” icon to generate your application using the most recent options you selected before (last output folder and file overwriting settings) without showing the options dialogs.
- **F3**: Show the project properties pane.
- **Shift + F3**: Show the application theme selector window.
- **F1**: Show/hide the context help pane.
- **F4**: Navigate between the project browser (left pane), the properties pane (right pane), and the project search box.

See also: Shortcut keys in generated applications

Automatic application uploader

As of AppGini 23.10, we introduced a new feature that makes it much easier to deploy (upload) your AppGini apps to your server. By clicking a single ‘Upload’ button, AppGini checks the changed files in your app and uploads them to your server. You no longer need to use external FTP, SSH or other upload tools, and you don’t have to worry about uploading the right files to the right folders.

Smart features of the automatic file uploader

- **Uploads only the files that have changed.** This means that if you make changes to your app that cause only a few files to change, the uploader will take care of detecting which files were changed and will only upload those files. This is much faster than uploading all files every time.
- **Uploads only the files that are needed.** The uploader will only upload the files that are needed to run your app. It will ignore files that are not needed, such as log files, the local `config.php` (which should not be uploaded to the server), ... etc.
- **Uploads files to the right folders.** The uploader will upload the files to the right folders on your server. For example, the file named `admin/index.php` in your app will be uploaded to the `admin` folder on your server. Missing folders will be created automatically as well. This saves you a lot of time trying to debug why your app is not working after uploading it.
- **Secure upload key.** The uploader uses a secret upload key to authenticate itself to your server. This key is created the first time you use the uploader. It’s a 32-character random string that is stored in AppGini config and in the `file-uploader.php` file. This all happens transparently to you, so you don’t have to worry about it. Just rest assured that any one trying to access the file uploader will not be able to do so without knowing the secret key.

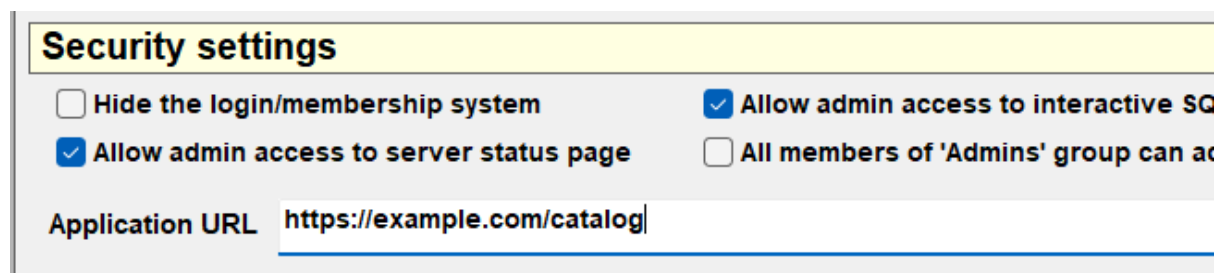
In addition, automatic file uploading only works over HTTPS. This is to prevent anyone from intercepting the upload key and using it to upload files to your server.

How to enable automatic file uploading

To enable automatic file uploading, you should:

1. **Set the application URL** under the Security settings section of the project properties. This is the URL that your users will use to access your app. You should provide the full URL of the homepage, including the protocol (`https://`) but without `index.php` at the end. For example:

`https://example.com/catalog`



Security settings

☐ Hide the login/membership system ☒ Allow admin access to interactive SQL

☒ Allow admin access to server status page ☐ All members of 'Admins' group can access

Application URL

Figure 46: Set the application URL

2. **Manually upload the generated file-uploader.php file to your server.** This only needs to be done once. You can do it by using any FTP or SSH tool. Once you upload it, you can use automatic uploading for all future changes. The `file-uploader.php` file is located in the home folder of your app. And it should be uploaded to the matching folder on your server. So, for the example application URL above, that file should be accessible at:

<https://example.com/catalog/file-uploader.php>

How to use automatic file uploading

After you've set the application URL and uploaded `file-uploader.php` to your server, you can use automatic file uploading by clicking the 'Upload' button in AppGini. This button can be found at the top of the project properties pane, next to the 'View files' button. It's only visible after you've generated your app.

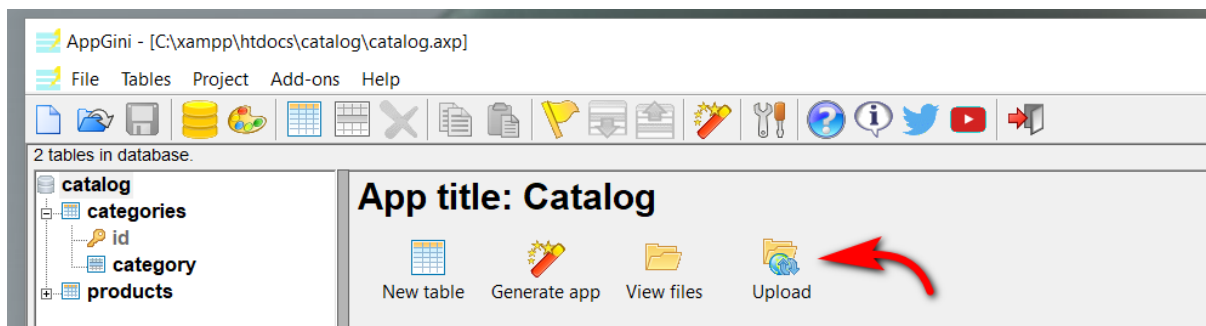


Figure 47: Upload button in the project properties pane

You can also find the 'Upload' button after generating your app in the status window:

PHP application generator status

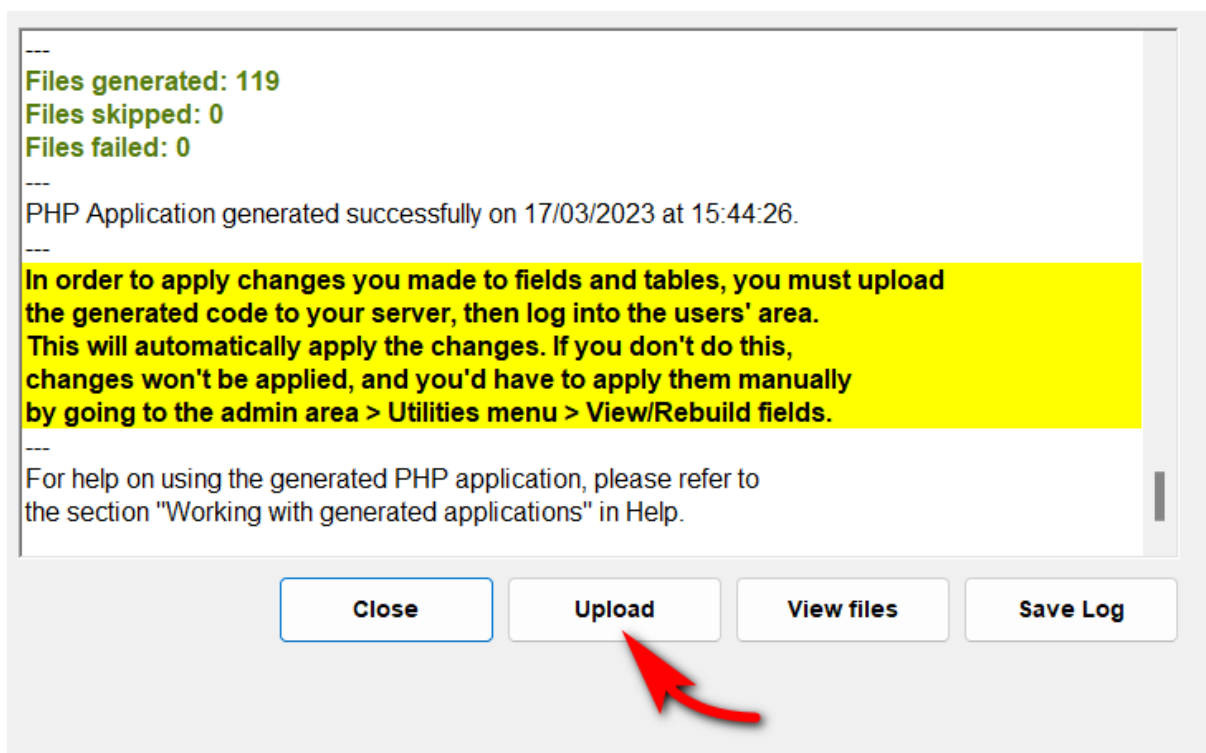


Figure 48: Upload button in status window

After clicking the 'Upload' button, you'll see a list of checks that AppGini performs before uploading.

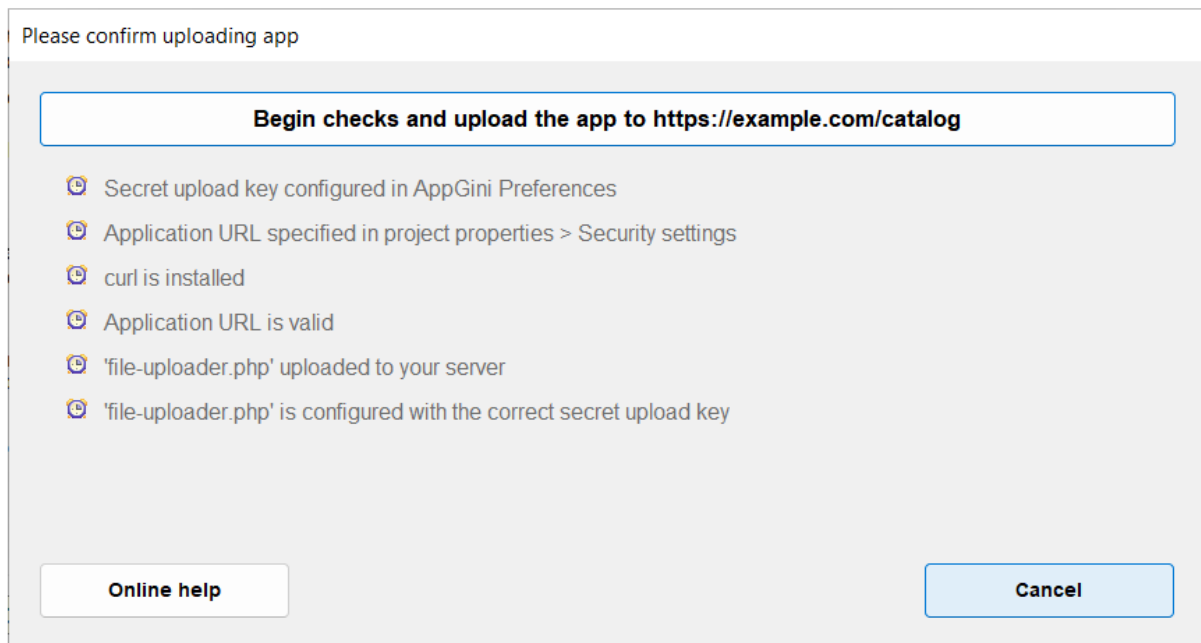


Figure 49: Checks performed before uploading apps in AppGini

Click the 'Begin checks and upload the app' button to start checks. If any checks fail, you'll see an error message:

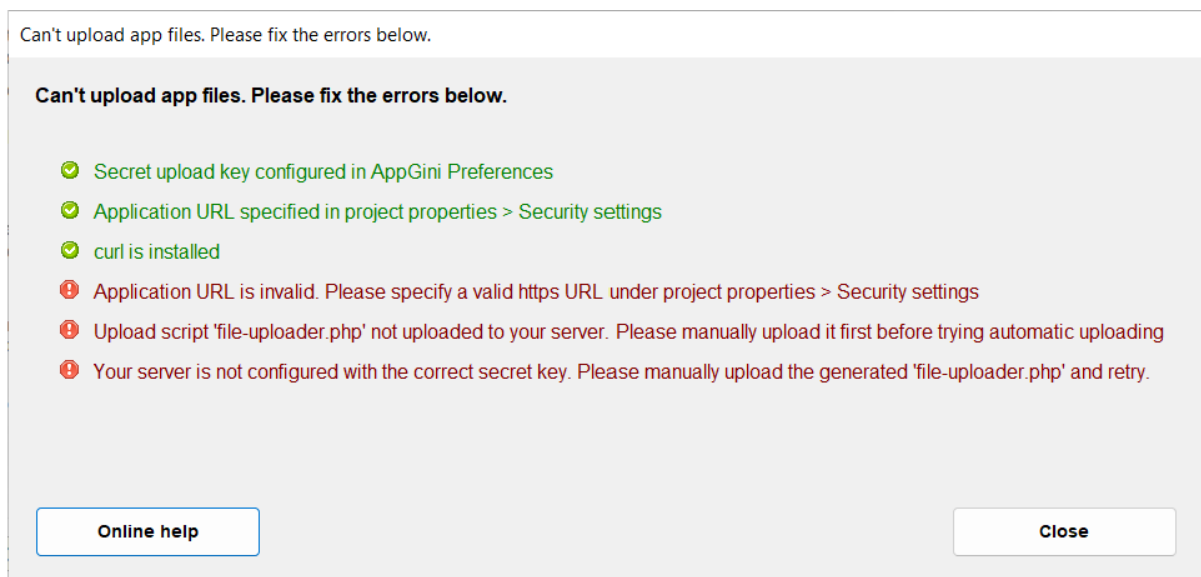


Figure 50: Upload checks failed

If all checks pass, AppGini will scan for file changes. This would take a couple of minutes or so, depending on the size of your app.

After that, AppGini will begin uploading only the changed files to your server. The upload progress window will show you the progress of the upload.

After the upload is complete, AppGini will show you how many files were uploaded, how many were skipped and how many failed, if any, along with a full list of files processed.

Troubleshooting

If you're having issues with automatic file uploading, please check the following:

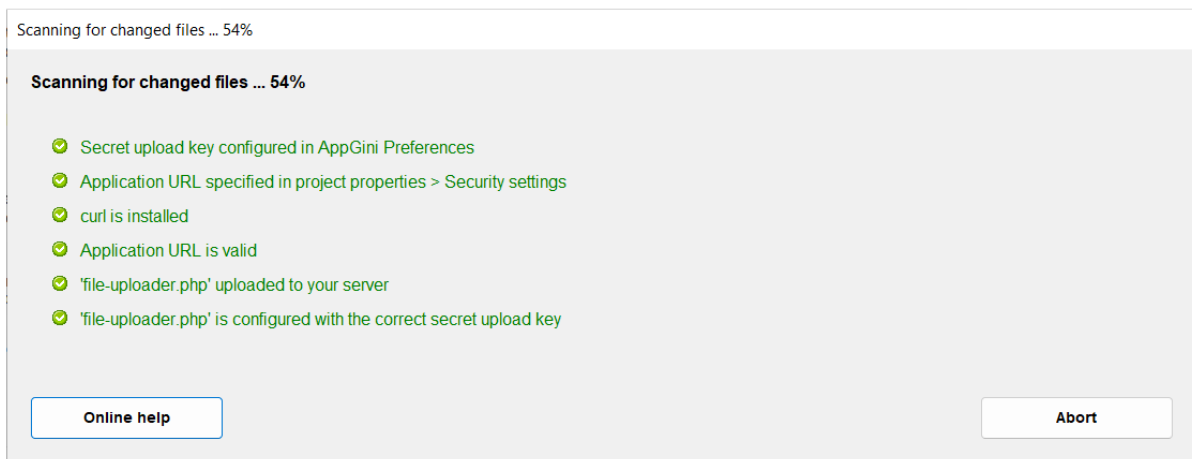


Figure 51: Scanning for changed files



Figure 52: Upload progress window

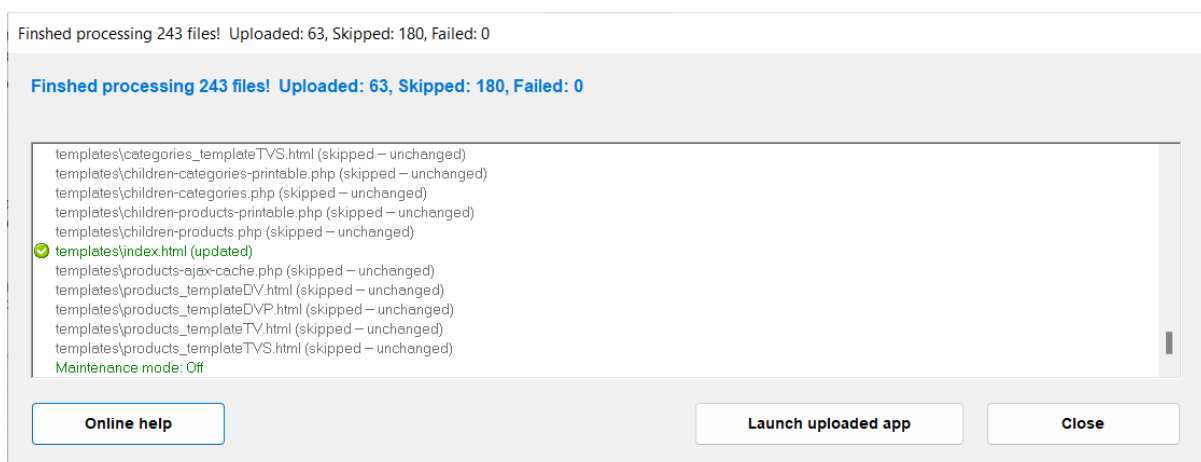


Figure 53: Upload results window

- **Have you set the application URL?** If you haven't set the application URL, the 'Upload' button will display an error. Please refer to the How to enable automatic file uploading section above for more information.
- **Have you uploaded file-uploader.php to your server?** If you haven't uploaded file-uploader.php to your server, the checks performed before uploading will fail. You can upload file-uploader.php to your server using any FTP or SSH tool. It should be uploaded to the home folder of your app on your server.
- **Is your application accessible over HTTPS?** Automatic file uploading only works over HTTPS. This is to prevent anyone from intercepting the upload key and using it to upload files to your server. Make sure your server has a valid, non-self-signed SSL certificate, and make sure it's not expired.
- **Make sure the secret upload key is correct.** The secret upload key can be retrieved from the AppGini preferences window. If it doesn't match the key in the file-uploader.php on your server, you can regenerate your app, then manually re-upload the new file-uploader.php file, overwriting the old one on the server.

Tip: You can view the secret upload key stored in the file-uploader.php file by opening it in a text editor. Line 2 contains the key, like so:

```
<?php
define('UPLOAD_KEY', '2DF5367D046FFE742277D04B107CF46B');
```

- **Is curl installed on your PC?** The automatic file uploader uses curl to upload files to your server. Curl is installed by default on modern Windows machines, Linux and MacOS. On older Windows PCs, you can download curl from the official curl website.
- **Do you have modsecurity or a similar web application firewall (WAF) installed on your server?** This might prevent the automatic file uploader from working. If you have a WAF installed on your server, you can try adding an exception for the file-uploader.php file to the WAF configuration. For modsecurity, you can try adding this code to a new file inside /etc/apache/mods-enabled/ (maybe name it appgini.conf) or similar, then restart apache:

```
<IfModule mod_security2.c>
    SecRule REQUEST_URI "/file-uploader.php$" id:300001,allow
</IfModule>
```

Hint: Check your server error logs to see if modsecurity is blocking requests to file-uploader.php or not.

- **Are you using Cloudflare?** Cloudflare is a great service for securing your website, but since it also acts as a web application firewall, it might block the automatic file uploader from working. You'll need to add an exception for the file-uploader.php file to Cloudflare's firewall rules.
- **Are folder permissions/ownership set correctly?** Make sure that the folder to which you're uploading your app and any subfolder are writable by the web server software (apache, nginx, .. etc) you're using. For example, on most apache setups on linux, the user that owns the app folders should be www-data

Security considerations

Automatic file uploading is a great feature, but it's important to understand the security implications of it. Here are some things to keep in mind:

- **The automatic file uploader uses HTTPS.** This is to prevent anyone from intercepting the upload key and using it to upload files to your server. Make sure your server has a valid, non-self-signed SSL certificate, and make sure it's not expired.
- **The automatic file uploader uses a secret upload key.** The secret upload key can be retrieved from the AppGini preferences window, under the 'App uploader' tab.

Make sure to keep this key secret. Anyone with access to this key can upload executable files to your server and compromise it. If you think your key has been compromised, you *must immediately*:

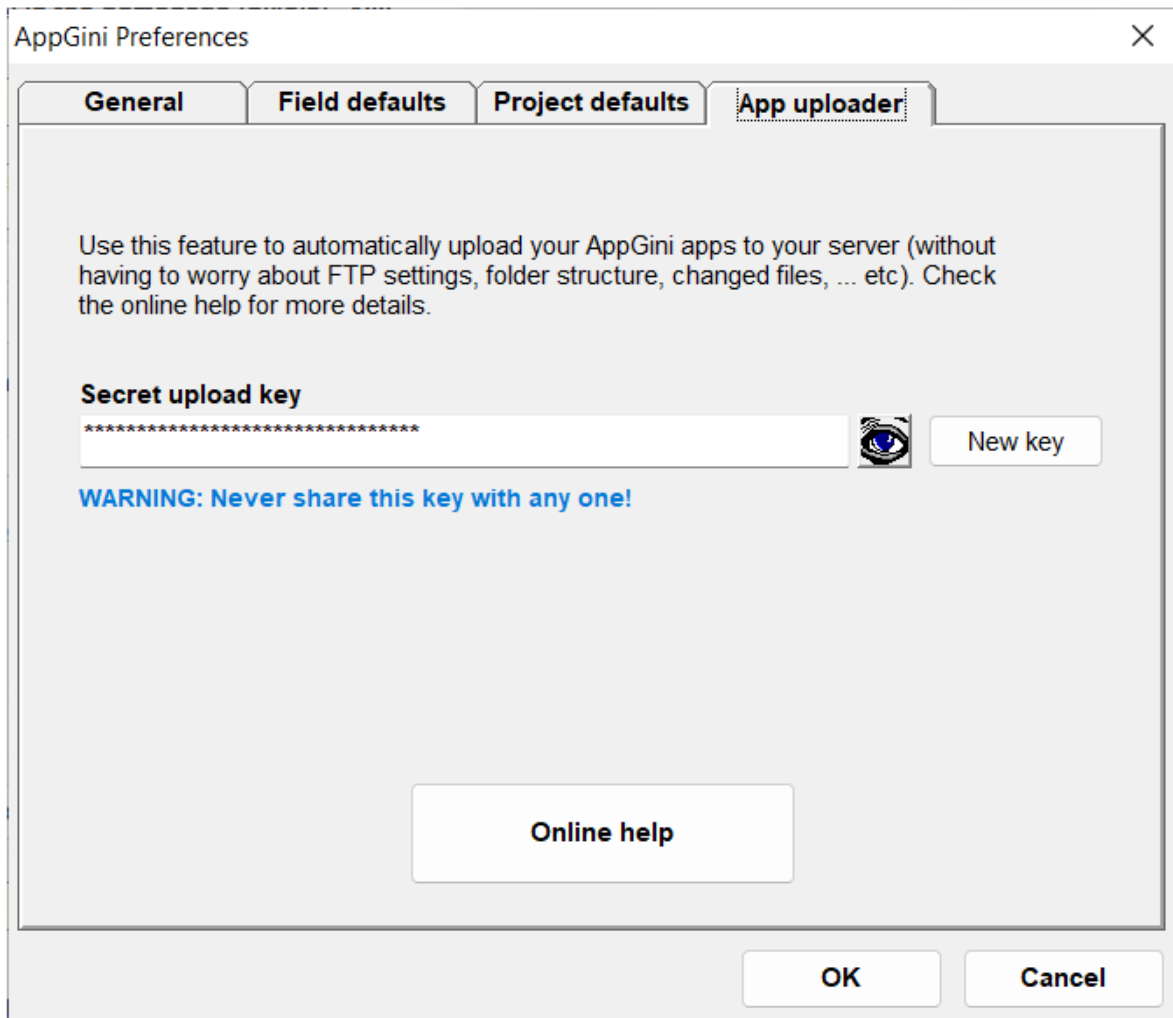


Figure 54: App uploader tab in AppGini preferences window

1. Remove the `file-uploader.php` file from your server.
2. Generate a new key from the AppGini preferences window.
3. Regenerate your app and upload the new `file-uploader.php` to your server.

We also recommend that you remove all app files from the server and use the automatic file uploader to re-upload them.

- During the upload process, the application is set to **maintenance mode**. This means that no one can access the app while it's being uploaded. After the upload is complete, the app is set back to normal mode.
- For tighter security, you can add a rule to your server firewall or to Cloudflare (if you're using it) to block access to the `file-uploader.php` file from all IP addresses except the one you're using to upload your app.

If you're using Apache, you can add this rule to your `.htaccess` file or your site's Apache configuration file:

```
<Files "file-uploader.php">
    Order allow,deny
    Deny from all
    Allow from 124.233.112.210
</Files>
```

Replace 124.233.112.210 with the external IP address of the PC you're using to upload your app.

For nginx, you can use this rule instead:

```
location ~* ^/file-uploader\.php$ {
    allow 124.233.112.210;
    deny all;
}
```

You could also specify a range of IP addresses in the above rules by using CIDR notation instead of a single IP address.

Setting the child record owner to match the owner of its parent record

As of AppGini 24.10, you can set the owner of a child record based on the owner of its parent record. This is useful in many scenarios, for example:

- You have a table of **projects** and another table of **tasks**. Each task belongs to a project. You want to set the owner of each task to the owner of its parent project.
- You have a table of **orders** and another table of **order_details**. Each order item belongs to an order. You want to set the owner of each order item to the owner of its parent order.

This is specially useful if you set the permissions of the child table to allow editing only by the owner of the record. If user Bob creates a project, then user Alice adds a task to that project, you might want the task to be owned by Bob so he can edit it.

To set the owner of a child record to the owner of its parent record, you need to do the following:

1. In AppGini, go to the table of the child records (e.g. **tasks** or **order_details** in the above examples).
2. Under the **Detail view settings** section, open the **Record owner** dropdown and select the lookup field that points to the parent table (e.g. **ProjectID** or **OrderID** in the above examples).

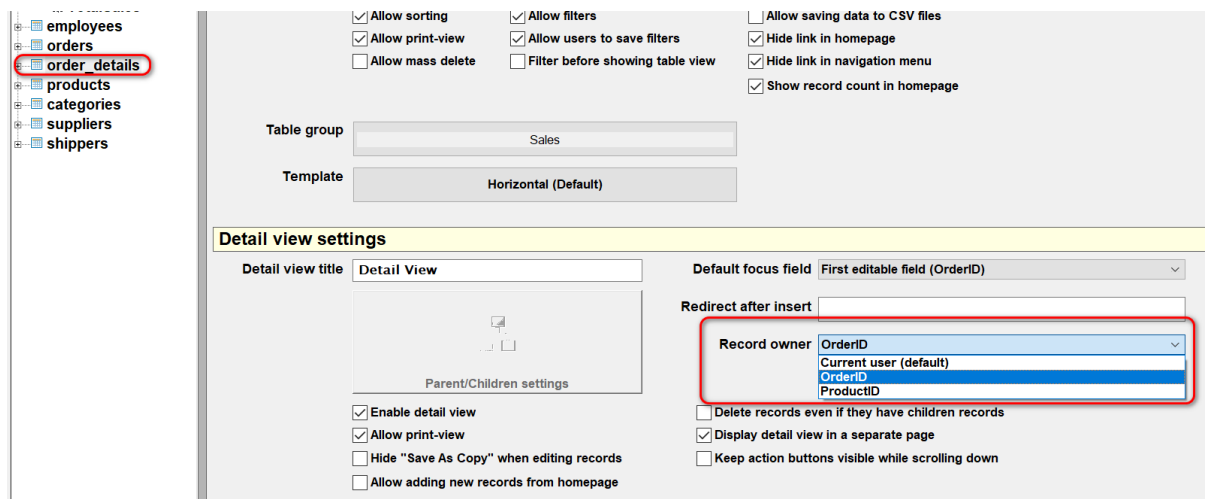


Figure 55: Record owner

3. Regenerate and upload your AppGini application.

Now, whenever an order item is created or edited, its owner user will be set to the owner of its parent order.

Updating the owner of existing child records

If you already have existing child records, the above setting will not apply to them unless you update each record manually. This can be a pain if you have many records. That's why we've also added a utility to mass update the owner of existing child records.

To use this utility, follow these steps:

1. Sign in as an administrator to your AppGini application.
2. Go to the admin area and open the **Utilities** menu.
3. Click on **Fix record owners**.

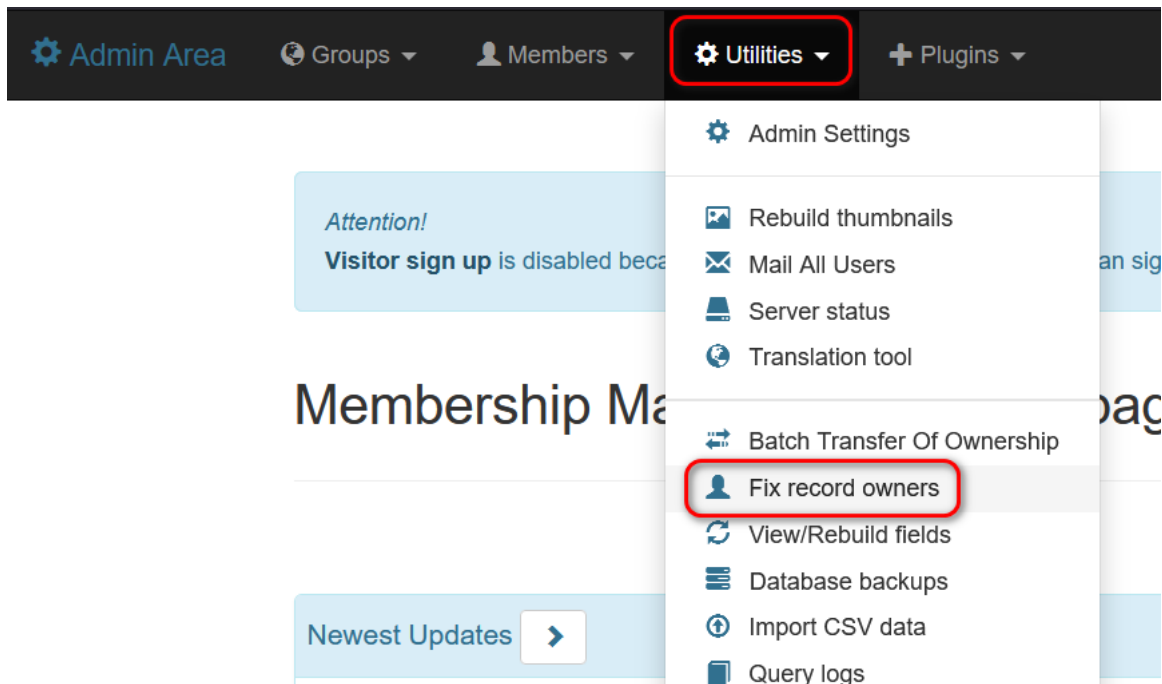


Figure 56: Utilities /> Fix record owners

4. This would open a page that lists all tables where the owner of child records can be updated. Click the **Start** button above the list of tables.
5. Leave the page open until the process is complete. This might take a while if you have many records.
6. Once the process is complete, the button would display the message **Done**. You can now close the page or navigate to another page.

A practical example: Set the country sales manager as the owner of all orders and order items of that country

Watch the video screencast

In this screencast, we are using the Northwind sample application. We configure the record owner of the **orders** table to be the **CustomerID** field, which is a lookup field pointing to the **customers** table. This means that the owner of each order would be the same as the owner of the customer record it belongs to.

We also configure the record owner of the **order_details** table to be the **OrderID** field, which in turn is a lookup field pointing to the **orders** table. This means that the owner of each order item would be the same as the owner of the order it belongs to.

This way, the owner of the customer record would be the owner of all orders and order items belonging to that customer. We'll then list all customers from Germany and set the sales manager of Germany as the owner of all German customers' records.


Fix record owners

This tool enables you to update the record owners for pre-existing records in tables that have record owners configured. This is useful when you have recently configured record owners for one or more tables and wish to apply the updated configuration to the existing records.

▶ Start


Number of runs

0 / 2

 Orders

0%

Records updated: 0

 Order Items

0%

Records updated: 0

Figure 57: Fix record owners page


Fix record owners

This tool enables you to update the record owners for pre-existing records in tables that have record owners configured. This is useful when you have recently configured record owners for one or more tables and wish to apply the updated configuration to the existing records.

✓ Done.


Number of runs

2 / 2

 Orders

Done.

Records updated: 0

 Order Items

Done.

Records updated: 0

Figure 58: Fix record owners page done

If a sales person creates an order for a German customer, the sales manager of Germany would become the owner of that order and all order items belonging to that order.

To apply this to existing German orders and order items, we use the **Fix record owners** utility to update the owner of all existing orders and order items.

Working with the generated web database application

After generating the PHP application based on your project, the next step is to upload the files to your server and set up the database. The generated files are saved to a folder you specify. Below is a screenshot of a folder containing files generated from a project.

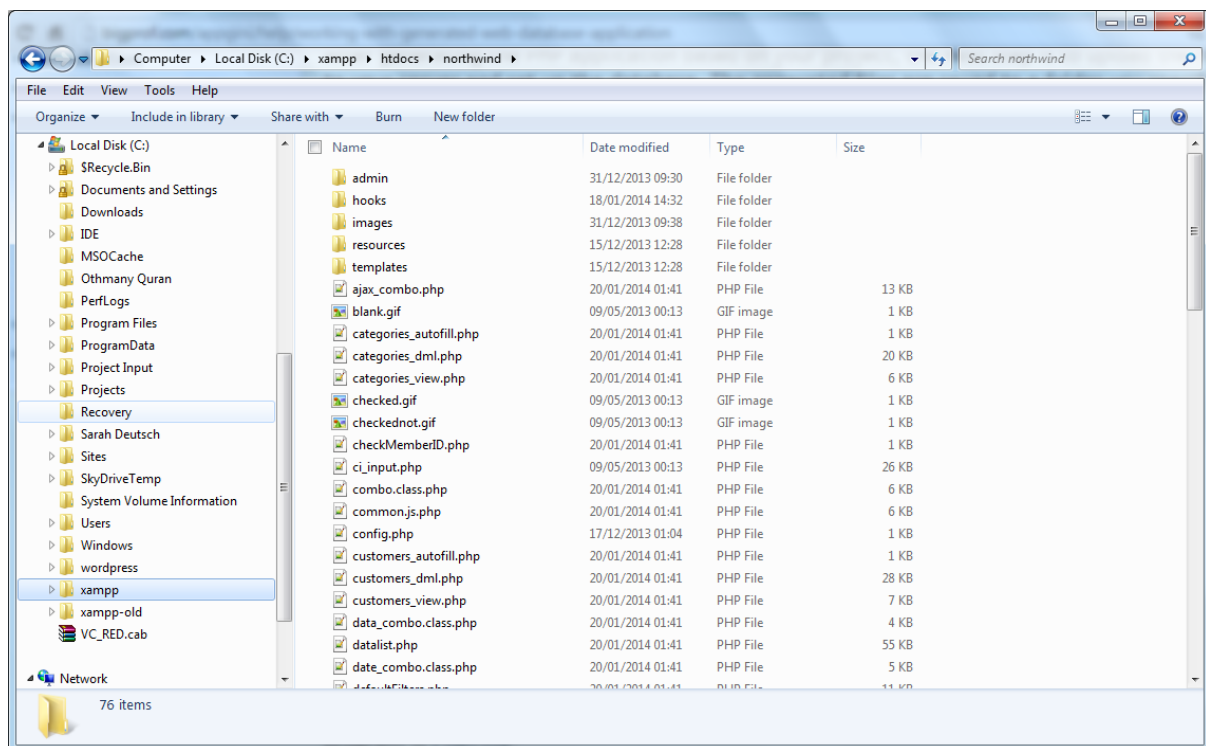


Figure 59: A folder containing files generated from an AppGini project

To upload the generated files, you should use an FTP client. A very good (and open source) program is FileZilla .

You should upload the entire folder to your web server. Make sure that your web server is properly configured to run .php files as PHP scripts (otherwise, they will probably be treated as text files and their entire source code will be displayed in the visitors' browsers.)

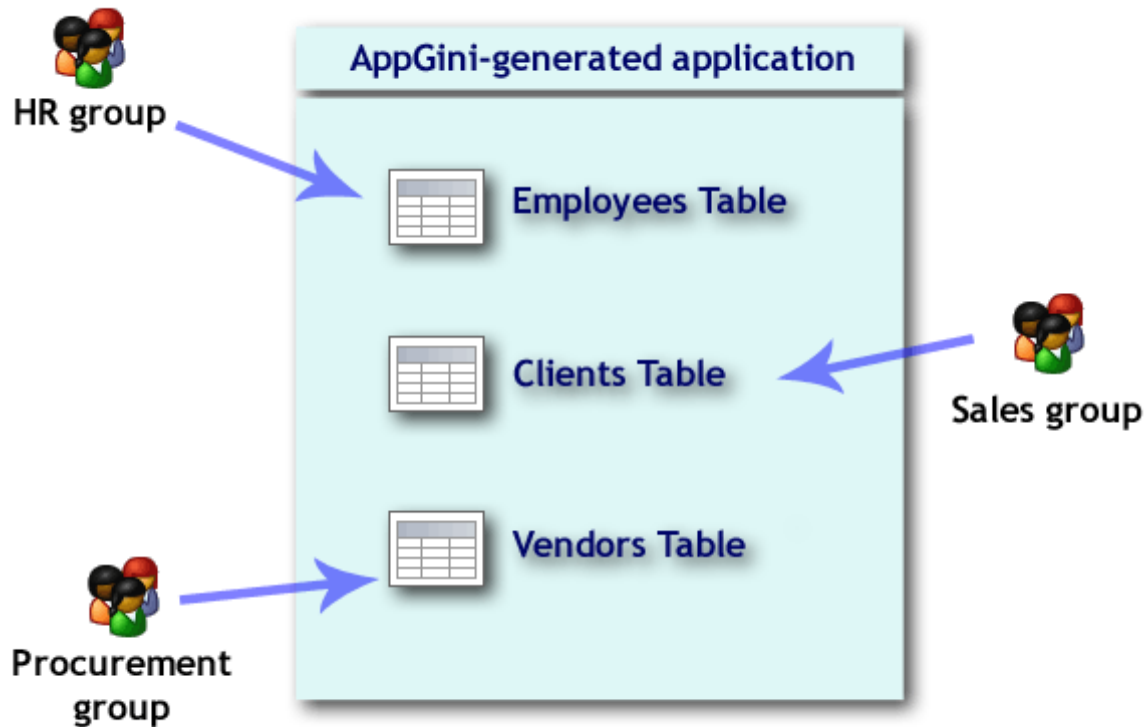
After you upload the files, you are ready to set up the database. Let's move on!

IMPORTANT Security note! Your database contains important information that you do not want any unauthorized person to mangle with ... So, only authorized users should have access to your database.

The generated AppGini app implements an advanced, yet intuitive, user management system. This system allows users to log into the generated application and have limited permissions that you (the admin) have full control of.

The admin has access to an admin area where he can define groups. Each group has its own permissions over each table in your application.

For example, let's say that you have created an application for storing clients' contacts, vendors' contacts, and employees' contacts. The admin can define a group called 'HR' which can view and edit only the employees' contacts, a group called 'Sales' which can view and edit only the clients' contacts, and a group called 'Procurement' which can view and edit only the vendors' contacts. Each group can have one or more members, and each member inherits his group's permissions. The following diagram explains this graphically.



If a user of the Sales group tries to access the Vendors table, he will not be permitted. If an anonymous user tries to access any table, he will not be permitted. If the admin changes the access permissions of a group, all members of that group will instantly be granted the new permissions (and denied the old ones.)

You can set the permissions of anonymous users in AppGini before file generation. And you can change them later from the admin area . Please be very careful with setting the anonymous permissions to avoid compromising your data.

A briefing of the generated files

For each table in your project, AppGini will generate several files. For example, in the above file list, the "categories" table has these files:

- `templates/categories_templateDV.html` This file contains the template that controls the layout of the detail view form of the table. This form is where users can enter new records or edit existing ones.
- `templates/categories_templateDVP.html` This file contains the template that controls the layout of the printer-friendly detail view form of the table.
- `templates/categories_templateTV.html` This file contains the template for displaying each record in the table view. The table view is a list of the records in the table.
- `templates/categories_templateTVS.html` This is the same as the `categories_templateTV.html`, except that it controls the template for the selected record only. When users click on a record in

the table view to select it, the selected record is highlighted in the table view, and its contents are displayed in the detail view for editing or deleting.

- **templates/children-categories.php** If categories table is displayed as a child of another table, this is the file used to format the child view.
- **categories_dml.php** This file contains the code that controls what happens on inserting a new record into the table, editing an existing record, or deleting a record.

This file also contains the **form()** function which controls the display of the detail view, using the **categories_templateDV.html** template file.

- **categories_autofill.php** If you have auto-fill lookup fields in your table, this file contains the code to populate these autofill fields. This file is called through an ajax request and sends javascript code to the browser.
- **categories_view.php** This is the controller page that welds all the above files together into a single page. You can control several display options and permissions in this page.

Setup

After uploading your PHP application files to a folder on your server, you can access it by pointing your browser to this URL:

`https://www.yourserver.com/path_to_appgini_generated_app`

Replace `www.yourserver.com` with your server name or IP, and `path_to_appgini_generated_app` with the path to your application's folder.

STEP 1 OF 3: Running the setup script

If you are running the generated application for the first time, you will see the following screen when you try to access the above url:

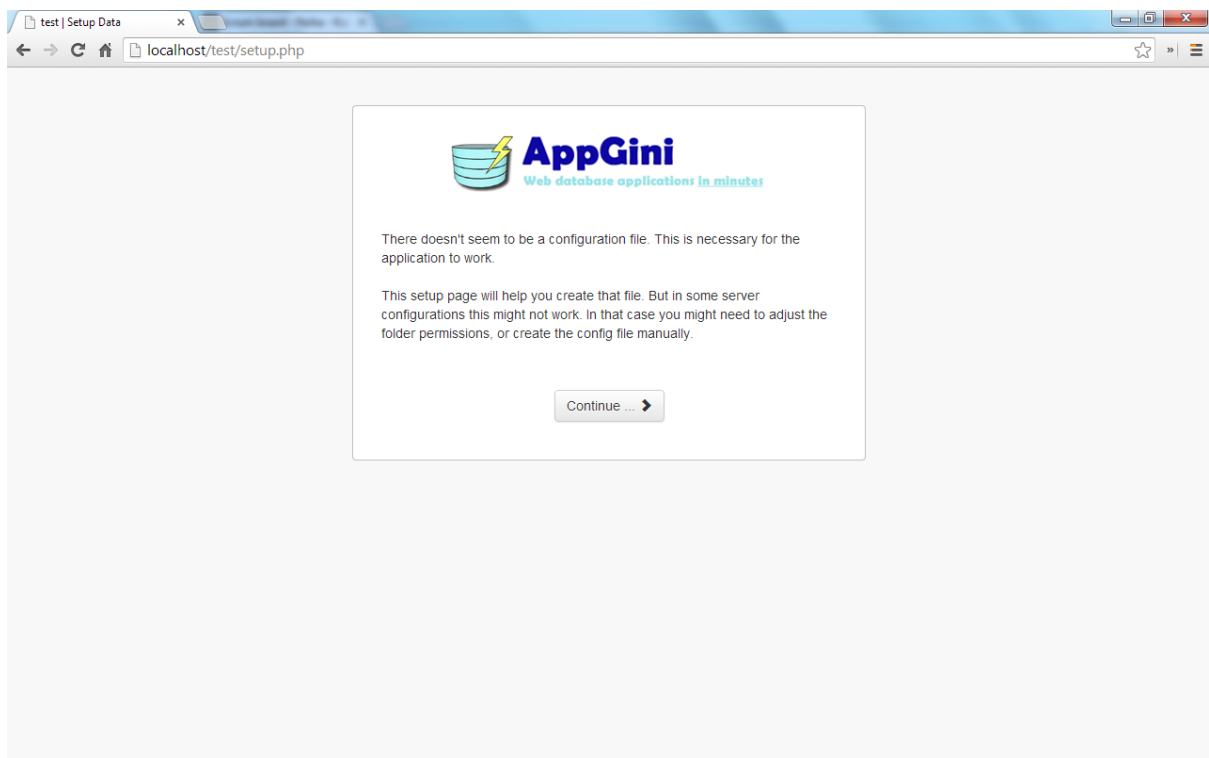


Figure 60: Setup step 1

Click on the “Continue” button to proceed. You should see a list of required information to prepare for the next step:

After clicking “Let’s go!”, you will be asked to provide the database login parameters, and create the admin account.

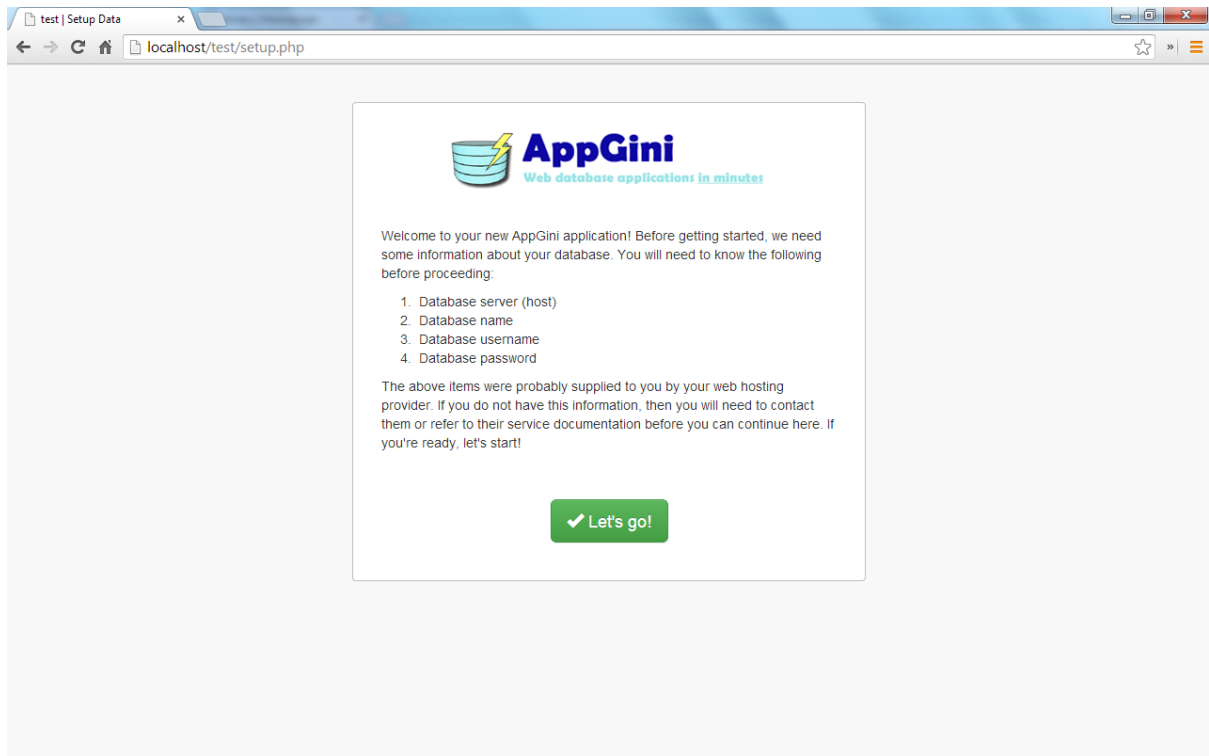


Figure 61: Setup step 2

test | Setup Data

localhost/test/setup.php?show-form=1

Setup Data

Database Information

MySQL server (host)

Database name

MySQL Username

MySQL password

Admin Information

Username

Email Address

Password
 Confirm Password

Submit

Next, the script will look for tables in the database with the names you specified in your project. It will attempt to create any table that doesn't already exist. You will see messages indicating whether the setup was successful or not, and a link to your PHP application's homepage. The home page is the file 'index.php' created in the application folder.

If you see error messages stating that the setup script can't create the database or any of the tables, make sure the database username and password you provided in the previous step have enough permissions to

allow you to define the database and its tables.

STEP 2 OF 3: Logging to the admin control panel

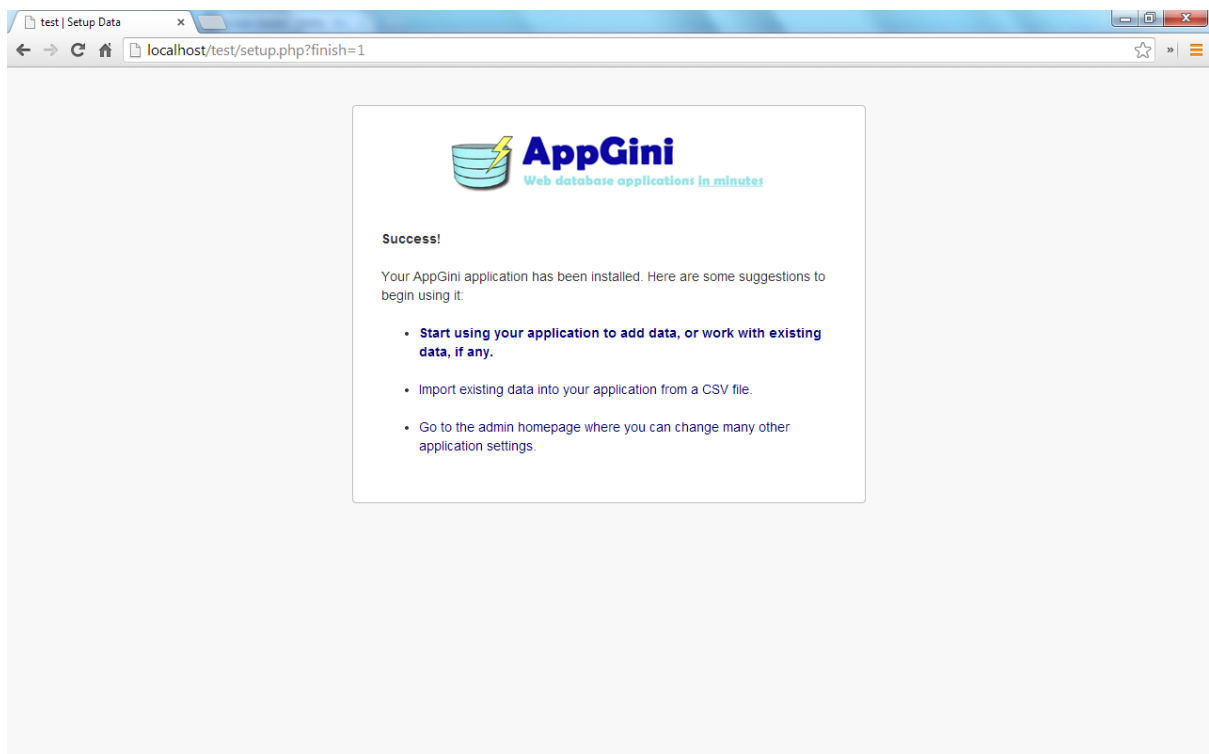
When you finish step 1 above and go to the home page, you'll see this screen:

Click on the "admin control panel" link. You should see the following screen:

Type the admin username and password and click "Sign In". The default username is admin and the default password is admin. In the next step, we shall change them.

STEP 3 OF 3: Changing the admin password

After you sign in as admin, you'll see the following page:



You should click on the "Admin Settings" link provided in the warning message above. This opens the following page, where you should type a new password and click "Save Changes":

Admin Settings

☒ Show tool tips as mouse moves over options

Admin username:

Admin password:

Confirm password:

Sender email:
Sender name and email are used in the 'To' field when sending email messages to groups or members.

Admin notifications:

- ☒ No email notifications to admin.
- ☐ Notify admin only when a new member is waiting for approval.
- ☐ Notify admin for all new sign-ups.

Sender name:

Members custom field 1:

Members custom field 2:

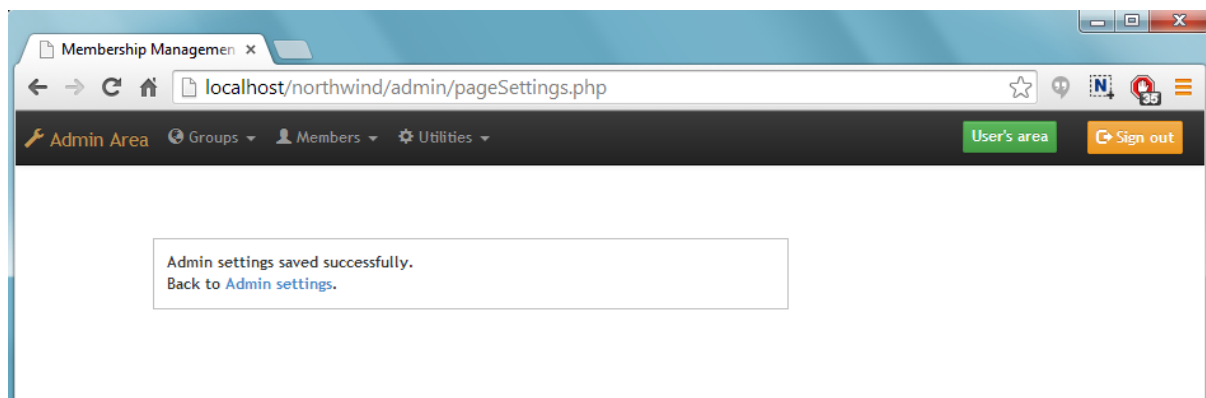
Members custom field 3:

Name of the anonymous group:

Name of the anonymous user:

[Save changes](#)

After saving the changes, you'll see this confirmation page:



If you'd like to have a look on the generated table pages, Click "Sign Out", and then click the "Go to user's area" link from the following page.

For more information about using the admin area and managing users, please refer to the admin interface section.

Working with tables and records

The application homepage

This page provides links to accessible tables in your application, depending on the group permissions of the logged user. We shall select the Customers Table.

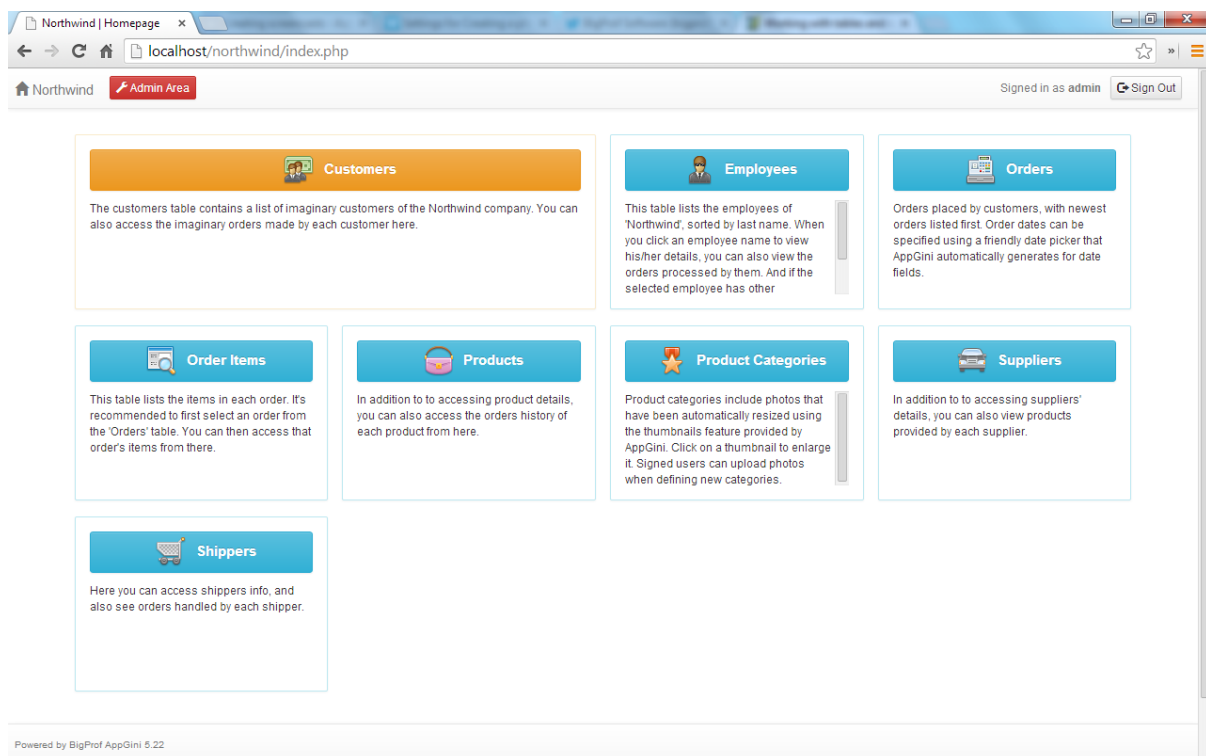


Figure 62: The application homepage

The Table View page

Here you can navigate data records of your table and edit, add and delete records. As shown in the figure below, the table view shows data in a table where records appear in rows, and each column represents a field of the table.

Column headers are specified in your AppGini project (as field captions). If you click on any of them, the table is sorted ascendingly by the clicked field. The figure below shows our table sorted by Country.

If you click again on the 'Country' header, it will be sorted descendingly. You can sort any field by simply clicking on its column header once or twice (to sort ascendingly or descendingly).

Northwind | Customers x

localhost/northwind/customers_view.php

Northwind Jump to ... Admin Area Signed in as admin Sign Out

Customers

Quick Search

Add New Print Preview Save CSV Filter Show All

<input type="checkbox"/>	Customer ID	Company Name	Contact Name	Address	City	Region	Country
<input type="checkbox"/>	ALFKI	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin		Germany
<input type="checkbox"/>	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.		Mexico
<input type="checkbox"/>	ANTON	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.		Mexico
<input type="checkbox"/>	AROUT	Around the Horn	Thomas Hardy	120 Hanover Sq.	London		United Kingdom
<input type="checkbox"/>	BSBEV	B's Beverages	Victoria Ashworth	Fauntleroy Circus	London		United Kingdom
<input type="checkbox"/>	BERGS	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå		Sweden
<input type="checkbox"/>	BLAUS	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim		Germany
<input type="checkbox"/>	BLONP	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg		France
<input type="checkbox"/>	BOLID	Bólido Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid		Spain
<input type="checkbox"/>	BONAP	Bon app'	Laurence Lebihan	12, rue des Bouchers	Marseille		France

Records 1 to 10 of 91

Previous Go to page: 1 Next

Powered by BigProf AppGini 5.30

Figure 63: The table view page

Northwind | Customers x

localhost/northwind/customers_view.php

Northwind Jump to ... Admin Area Signed in as admin Sign Out

Customers

Quick Search

Add New Print Preview Save CSV Filter Show All

<input type="checkbox"/>	Customer ID	Company Name	Contact Name	Address	City	Region	Country 📄
<input type="checkbox"/>	CACTU	Cactus Comidas para llevar	Patricio Simpson	Cerrito 333	Buenos Aires		Argentina
<input type="checkbox"/>	RANCH	Rancho grande	Sergio Gutiérrez	Av. del Libertador 900	Buenos Aires		Argentina
<input type="checkbox"/>	OCEAN	Océano Atlántico Ltda.	Yvonne Moncada	Ing. Gustavo Moncada 8585 Piso 20-A	Buenos Aires		Argentina
<input type="checkbox"/>	PICCO	Piccolo und mehr	Georg Pipp	Geislweg 14	Salzburg		Austria
<input type="checkbox"/>	ERNSH	Ernst Handel	Roland Mendel	Kirchgasse 6	Graz		Austria
<input type="checkbox"/>	MAISD	Maison Dewey	Catherine Dewey	Rue Joseph-Bens 532	Bruxelles		Belgium
<input type="checkbox"/>	SUPRD	Suprêmes délices	Pascale Cartrain	Boulevard Tirou, 255	Charleroi		Belgium
<input type="checkbox"/>	FAMIA	Familia Arquibaldo	Aria Cruz	Rua Orós, 92	São Paulo	SP	Brazil
<input type="checkbox"/>	QUEDE	Que Delícia	Bernardo Batista	Rua da Panificadora, 12	Rio de Janeiro	RJ	Brazil
<input type="checkbox"/>	GOURL	Gourmet Lanchonetes	André Fonseca	Av. Brasil, 442	Campinas	SP	Brazil

Records 1 to 10 of 91

Previous Go to page: 1 Next

Powered by BigProf AppGini 5.30

Figure 64: Table sorted by Country, ascendingly

The Detail View

The detail view page can be used to enter data of a new record. To do so, click the "ADD NEW" button. In addition, when you click on any record in the table, its data is displayed in the Detail View for viewing, editing, deleting, printing, and/or copying to a new record. The exact functions enabled depend on the permissions of the logged user.

The screenshot displays the 'Detail View' for an order with ID 11077. The form contains the following fields and values:

Field	Value
Order ID	11077
Customer	Rattlesnake Canyon Grocery
Employee	Davolio, Nancy
Order Date	December 5, 2016
Required Date	January 5, 2017
Shipped Date	
Ship Via	United Package
Freight	8.53
Ship Name	Rattlesnake Canyon Grocery
Ship Address	2817 Milton Dr.
Ship City	Albuquerque

On the right side of the form, there are several action buttons: a green 'Save' button with a checkmark, a 'Back' button, a 'Print Preview' button, a 'Delete' button, and a printer icon. The top navigation bar includes the 'Northwind' logo, a 'Jump to ...' dropdown, 'Signed in as admin', and the 'Orders' section header.

Figure 65: The detail view page

To sum up, the table view allows you to control and manage data of the table by allowing you to insert new records, edit and delete records; sort and navigate data, filter data or move to any other table. All this can be done in one page in an easy to understand manner.

Remember that you can control the appearance of the table view through Project Styles in AppGini. You can also control table view and detail view layouts by editing the generated template files .

Working with filters

Filters allow users to have advanced control over data display. The filters button brings a filters page as shown below.

Customers Filters

	Filtered field	Comparison Operator	Comparison Value	
Filter 01	<input type="text"/>	<input type="text"/>	<input type="text"/>	
Filter 02	<input type="text"/>	<input type="text"/>	<input type="text"/>	
Filter 03	<input type="text"/>	<input type="text"/>	<input type="text"/>	
Filter 04	<input type="text"/>	<input type="text"/>	<input type="text"/>	

Order by

Order by	<input type="text"/>	<input type="text"/>
Then by	<input type="text"/>	<input type="text"/>
Then by	<input type="text"/>	<input type="text"/>
Then by	<input type="text"/>	<input type="text"/>

Records to display

☐ Only your own records
☐ All records owned by your group
☒ All records

Powered by BigProf AppGini 5.40

Figure 66: Filters page

For example, if we wish to find all customers from France, Germany or Mexico, whose contact names begin with A, M or P, the filters would look like this:

If a filter begins with 'And' it means the condition must be fulfilled, and if it begins with 'Or' then the condition is optional. You can use % (percentage sign) and _ (underscore) in comparison values when the comparison operator is 'Like' or 'Not Like'. % means any number of characters and _ means any single character.

There are several comparison operators available for filters, the following drop-down from the filters page shows them all.

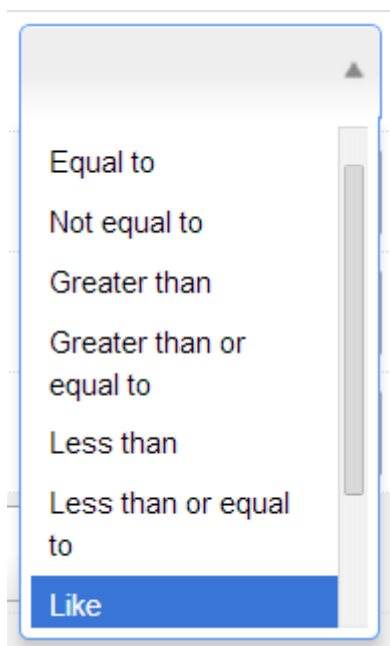
Northwind Jump to ... Admin Area Signed in as admin Sign Out

Customers Filters

	Filtered field	Comparison Operator	Comparison Value	
Filter 01	Country	Equal to	France	
Filter 02	Or	Country	Germany	
Filter 03	Or	Country	Mexico	
Filter 04				
And				
Filter 05	Company Name	Like	A%	
Filter 06	Or	Company Name	M%	
Filter 07	Or	Company Name	P%	
Filter 08				
Order by				
	Order by			
	Then by			
	Then by			
	Then by			
Records to display				
<input type="radio"/> Only your own records <input type="radio"/> All records owned by your group <input checked="" type="radio"/> All records				
<input checked="" type="button" value="Apply filters"/> <input type="button" value="Save and apply filters"/> <input type="button" value="Cancel"/>				

Powered by BigProf AppGini 5.40

Figure 67: Filters for customers from France, Germany or Mexico, whose contact names begin with A, M or P



To apply filters to the table view after specifying them, simply click the "Apply filters" button.

Related screencasts

Saving filters for use later

Performing advanced search queries in AppGini using filters

The admin interface

AppGini allows you to create member accounts and control the privileges of members. For each table in your application, you can control whether members can add new records, edit existing ones, and/or delete records. Moreover, you can control which records a member can edit and/or delete: only his own records (records added by the member himself), or his group's records (records added by any member of the group to which our member belongs), or all records entered by him and any other member of any group.

Member groups

To make administration of members easier, AppGini allows you to create groups and assign each member to a group. Thus, instead of assigning privileges to each individual member separately, you assign privileges to a group. All members of the group are then automatically assigned these privileges.

Sample scenario: A content publishing application

For example, if you are developing a content publishing application, you might create an authors group that has the privilege to add new records to the articles table. Each member of the authors group can edit his own records (articles), but not the articles of other members (authors).

You would also create an editors group. Members of this group can edit any record in the articles table, but are not allowed to delete or add records.

Finally, you might create a subscribers group. Members of this group can only read articles (that is, view records of the articles table), but not edit, delete or add records.

Accessing the admin homepage

The admin interface allows you to define groups and their privileges, approve and ban members, send email notifications to groups or individual members, plus other administrative tasks.

You can access the admin interface by signing in as an admin user, then clicking the 'Admin area' link at the top of the page. This will take you to the admin homepage, similar to the one below.

After signing in, you'll see the admin homepage, which provides a quick review of latest events: newest members, most active members, newest records and updates, plus links to all admin tools.

Managing groups

To view available groups, click the 'View Groups' link on the top of the admin homepage. This would display a page similar to this one below.

If you click the "Edit" icon to the left of a group, you can edit the group's details and permissions (privileges). This will open a page similar to this one below.

Membership Management Homepage

<div>Newest Updates</div> <table> <tr> <td>12/22/2012, 08:55 am</td><td>ALFKI, Alfreds ...</td></tr> <tr> <td>12/06/2012, 10:01 pm</td><td>51, Manjimup Dr ...</td></tr> <tr> <td>12/05/2012, 08:13 am</td><td>2, Dr., 5220820 ...</td></tr> <tr> <td>12/04/2012, 12:59 pm</td><td>18, Carnarvon T ...</td></tr> <tr> <td>12/04/2012, 12:56 pm</td><td>39, Chartreuse ...</td></tr> </table>	12/22/2012, 08:55 am	ALFKI, Alfreds ...	12/06/2012, 10:01 pm	51, Manjimup Dr ...	12/05/2012, 08:13 am	2, Dr., 5220820 ...	12/04/2012, 12:59 pm	18, Carnarvon T ...	12/04/2012, 12:56 pm	39, Chartreuse ...	<div>Newest Entries</div> <table> <tr> <td>08/09/2012, 06:02 am</td><td>10402, ERNSH, 8 ...</td></tr> <tr> <td>08/09/2012, 06:02 am</td><td>10658, QUICK, 4 ...</td></tr> <tr> <td>08/09/2012, 06:02 am</td><td>10914, QUEEN, 6 ...</td></tr> <tr> <td>08/09/2012, 06:02 am</td><td>93, 10283, 19, ...</td></tr> <tr> <td>08/09/2012, 06:02 am</td><td>349, 10380, 53, ...</td></tr> </table>	08/09/2012, 06:02 am	10402, ERNSH, 8 ...	08/09/2012, 06:02 am	10658, QUICK, 4 ...	08/09/2012, 06:02 am	10914, QUEEN, 6 ...	08/09/2012, 06:02 am	93, 10283, 19, ...	08/09/2012, 06:02 am	349, 10380, 53, ...
12/22/2012, 08:55 am	ALFKI, Alfreds ...																				
12/06/2012, 10:01 pm	51, Manjimup Dr ...																				
12/05/2012, 08:13 am	2, Dr., 5220820 ...																				
12/04/2012, 12:59 pm	18, Carnarvon T ...																				
12/04/2012, 12:56 pm	39, Chartreuse ...																				
08/09/2012, 06:02 am	10402, ERNSH, 8 ...																				
08/09/2012, 06:02 am	10658, QUICK, 4 ...																				
08/09/2012, 06:02 am	10914, QUEEN, 6 ...																				
08/09/2012, 06:02 am	93, 10283, 19, ...																				
08/09/2012, 06:02 am	349, 10380, 53, ...																				
<div>Top Members</div> <table> <tr> <td>admin</td><td>3202 records</td></tr> </table>	admin	3202 records	<div>Members Stats</div> <table> <tr> <td>Total groups</td><td>3</td></tr> <tr> <td>Active members</td><td>3</td></tr> <tr> <td>Members awaiting approval</td><td>0</td></tr> <tr> <td>Banned members</td><td>0</td></tr> <tr> <td>Total members</td><td>3</td></tr> </table>	Total groups	3	Active members	3	Members awaiting approval	0	Banned members	0	Total members	3								
admin	3202 records																				
Total groups	3																				
Active members	3																				
Members awaiting approval	0																				
Banned members	0																				
Total members	3																				

Figure 68: “The admin homepage of an AppGini application”

Groups

Add Group

Search groups		Find	Reset		
Group	Description	Members count			
anonymous	Anonymous group created automatically on 2015-12-07	1			
Admins	Admin group created automatically on 2015-12-07	2			
Sales		2			
HR		1			

Previous
Displaying groups 1 to 4 of 4
Next

Key:
Edit group details and permissions.
Delete group
Add a new member to group.
View all data records entered by the group's members.

List all members of a group.
Send an email message to all members of a group.

Figure 69: Managing user groups in an AppGini application

Admin Area
Groups
Members
Utilities
User's area
Sign out

Edit Group 'Admins'

Back to groups
View group members
Add a new member to group.
View group records

☒ Show tool tips as mouse moves over options

Group name
Admins

If you name the group 'anonymous', it will be considered the anonymous group that defines the permissions of guest visitors that do not log into the system.

Description
Admin group created automatically on 2015-12-07

Allow visitors to sign up?
☒ No. Only the admin can add users.
☐ Yes, and the admin must approve them.
☐ Yes, and automatically approve them.

Save changes

Scrolling down the group editing page, you'll see the group's permissions for each table. If you pass your mouse pointer over any item in the permissions section, you'll see a detailed description of what it means.

Admin Area
Groups
Members
Utilities
Save changes
Sign out

Edit Group 'Sales'

Back to groups
View group members
Add a new member to group.
View group records

☒ Show tool tips as mouse moves over options

Group name
Sales

If you name the group 'anonymous', it will be considered the anonymous group that defines the permissions of guest visitors that do not log into the system.

Description

Allow visitors to sign up?
☐ No. Only the admin can add users.
☐ Yes, and the admin must approve them.
☒ Yes, and automatically approve them.

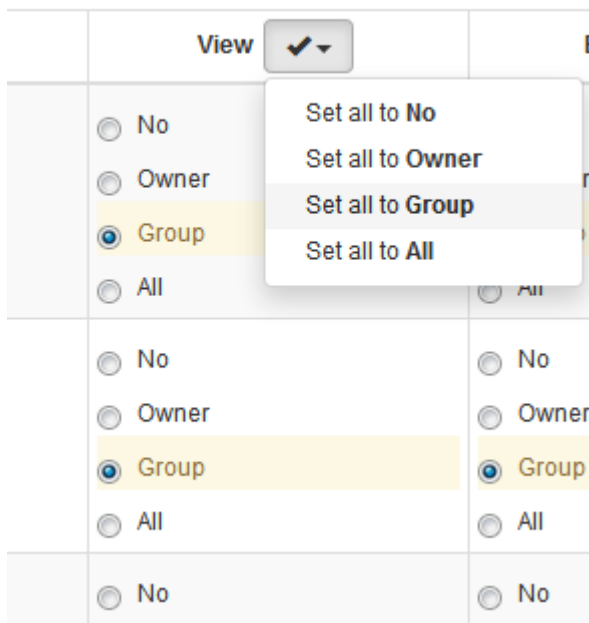
Save changes

Table permissions for this group

Table	Insert	View	Edit	Delete
Customers	<input checked="" type="checkbox"/>	<input type="radio"/> No <input type="radio"/> Owner <input checked="" type="radio"/> Group <input type="radio"/> All	<input type="radio"/> No <input type="radio"/> Owner <input checked="" type="radio"/> Group <input type="radio"/> All	<input type="radio"/> No <input checked="" type="radio"/> Owner <input type="radio"/> Group <input type="radio"/> All
Employees	<input checked="" type="checkbox"/>	<input type="radio"/> No <input type="radio"/> Owner <input checked="" type="radio"/> Group <input type="radio"/> All	<input type="radio"/> No <input type="radio"/> Owner <input checked="" type="radio"/> Group <input type="radio"/> All	<input type="radio"/> No <input checked="" type="radio"/> Owner <input type="radio"/> Group <input type="radio"/> All
Orders	<input checked="" type="checkbox"/>	<input type="radio"/> No <input type="radio"/> Owner <input checked="" type="radio"/> Group <input type="radio"/> All	<input type="radio"/> No <input type="radio"/> Owner <input checked="" type="radio"/> Group <input type="radio"/> All	<input type="radio"/> No <input checked="" type="radio"/> Owner <input type="radio"/> Group <input type="radio"/> All
Order Items	<input checked="" type="checkbox"/>	<input type="radio"/> No <input type="radio"/> Owner <input checked="" type="radio"/> Group <input type="radio"/> All	<input type="radio"/> No <input type="radio"/> Owner <input checked="" type="radio"/> Group <input type="radio"/> All	<input type="radio"/> No <input checked="" type="radio"/> Owner <input type="radio"/> Group <input type="radio"/> All
Products	<input checked="" type="checkbox"/>	<input type="radio"/> No <input type="radio"/> Owner <input checked="" type="radio"/> Group <input type="radio"/> All	<input type="radio"/> No <input type="radio"/> Owner <input checked="" type="radio"/> Group <input type="radio"/> All	<input type="radio"/> No <input checked="" type="radio"/> Owner <input type="radio"/> Group <input type="radio"/> All
Product Categories	<input checked="" type="checkbox"/>	<input type="radio"/> No <input type="radio"/> Owner <input checked="" type="radio"/> Group <input type="radio"/> All	<input type="radio"/> No <input type="radio"/> Owner <input checked="" type="radio"/> Group <input type="radio"/> All	<input type="radio"/> No <input checked="" type="radio"/> Owner <input type="radio"/> Group <input type="radio"/> All
Suppliers	<input checked="" type="checkbox"/>	<input type="radio"/> No <input type="radio"/> Owner <input checked="" type="radio"/> Group <input type="radio"/> All	<input type="radio"/> No <input type="radio"/> Owner <input checked="" type="radio"/> Group <input type="radio"/> All	<input type="radio"/> No <input checked="" type="radio"/> Owner <input type="radio"/> Group <input type="radio"/> All
Shippers	<input checked="" type="checkbox"/>	<input type="radio"/> No <input type="radio"/> Owner <input checked="" type="radio"/> Group <input type="radio"/> All	<input type="radio"/> No <input type="radio"/> Owner <input checked="" type="radio"/> Group <input type="radio"/> All	<input type="radio"/> No <input checked="" type="radio"/> Owner <input type="radio"/> Group <input type="radio"/> All

Save changes

In AppGini 5.80 and above, the buttons with checkmarks to the right of each permission allow you to apply the same permission value to all tables in one step. Clicking that button opens a menu, as shown at the right. If you click on *Set all to **Group***, for example, all tables will have the 'View' permission set to 'Group'.



To define a new group, open the *Groups* menu at the top of any admin page, and click the *Add Group* command. This will open a page similar to the group editing page but with empty fields for you to fill.

Managing members

To view available members, click the 'View Members' link on the top of the admin homepage. This would display a page similar to this one below.

Admin Area
Groups
Members
Utilities
User's area
Sign out

Members
Add New Member

Search members
in All fields
Group
Status Any
Find
Reset

Username	Group	Sign up date	Full Name	Address	City	State	Status	
admin	Admins	12/07/2015					Active	Edit Delete Ban Message
guest	anonymous	12/07/2015					Active	Edit Delete Ban Message
john	Admins	12/16/2016					Active	Edit Delete Ban Message
isa	Sales	12/16/2016					Active	Edit Delete Ban Message
carola	Sales	12/16/2016					Active	Edit Delete Ban Message
linda	HR	12/18/2016					Active	Edit Delete Ban Message

Previous
Displaying members 1 to 6 of 6
Next

Key:
[Edit member details](#)
[Delete member](#)
[Activate new/banned member.](#)
[Ban \(suspend\) member.](#)
[View all data records entered by member.](#)
[Send an email message to member.](#)

If you click the "Edit" icon to the left of a member, you can edit the member's details. This will open a page similar to this one below.

Admin Area
Groups
Members
Utilities
User's area
Sign out

Edit Member john
Back to members
View member's records
Send message to member

Member usernamejohn

Password

Type a password only if you want to change this member's password. Otherwise, leave this field empty.

Confirm password

Emailjohn@gmail.com

GroupAdmins

☒ Approved?
☐ Banned?

Full NameJohn Peterson

Address9695 Nichols St

CityEl Dorado

StateAR 71730

Comments

Note that AppGini allows you as an admin to ban (suspend) members temporarily. A banned member will not be able to sign in. You can unban him at any time later.

Managing records

The admin interface allows you to view all records entered by any member or group. Click the 'View Members' Records' link on the top of the admin homepage. This will display a page similar to the one below.

Admin Area
Groups
Members
Utilities
User's area
Sign out

Data Records

Group	Member username	Show records from	Sort records by	Find	Reset
	admin	All tables	Date created		
Username	Group	Table	Created	Modified	Data
admin	Admins	categories	12/24/2016, 02:16 pm	12/24/2016, 02:16 pm	1, 86051600_1344483927.jpg, Beverages, Soft drinks, coffees, teas, beers, and al ...
admin	Admins	categories	12/24/2016, 02:16 pm	12/24/2016, 02:16 pm	2, 24242400_1344483908.jpg, Condiments, Sweet and savory sauces, relishes, sprea ...
admin	Admins	categories	12/24/2016, 02:16 pm	12/24/2016, 02:16 pm	3, 91786600_1344483888.jpg, Confections, Desserts, candies, and sweet breads ...
admin	Admins	categories	12/24/2016, 02:16 pm	12/24/2016, 02:16 pm	4, 16103200_1344483866.jpg, Dairy Products, Cheeses ...
admin	Admins	categories	12/24/2016, 02:16 pm	12/24/2016, 02:16 pm	5, 57976100_1344483796.jpg, Grains/Cereals, Breads, crackers, pasta, and cereal ...
admin	Admins	categories	12/24/2016, 02:16 pm	12/24/2016, 02:16 pm	6, 21602000_1344483775.jpg, Meat/Poultry, Prepared meats ...
admin	Admins	categories	12/24/2016, 02:16 pm	12/24/2016, 02:16 pm	7, 26914200_1344483754.jpg, Produce, Dried fruit and bean curd ...
admin	Admins	categories	12/24/2016, 02:16 pm	12/24/2016, 02:16 pm	8, 13835000_1344483348.png, Seafood, Seaweed and fish ...
admin	Admins	logs	12/15/2016, 06:53 am	12/15/2016, 06:53 am	1, ::1, 1481138329, User login failed. Username: admin. Password: 1234556 ...
admin	Admins	suppliers	09/08/2016, 11:51 am	09/08/2016, 11:51 am	1, Exotic Liquids, Charlotte Cooper, Purchasing Manager, 49 Gilbert St., London, ...

Displaying records 1 to 10 of 3205
Next

If you click the 'Edit' icon to the left of any record, you can view all the data in that record, and you can also edit the record ownership. This will open a page similar to the one below.

Edit Record Ownership

Owner group

Sales

>

Owner member

demo

>

If you want to switch ownership of this record to a member of another group, you must change the owner group and save changes first.

Save changes

Record created on 08/09/2012, 06:02 am

Record modified on 11/22/2018, 06:10 pm

Table customers

Record data

Print
Edit

Field name	Value
CustomerID	BLAUS
CompanyName	Blauer See Delikatessen
ContactName	Hanna Moos
ContactTitle	Sales Representative
Address	Forsterstr. 57
City	Mannheim
Region	
PostalCode	68306
Country	Germany
Phone	0621-08460
Fax	0621-08924
NumOrders	247

If you want to change the ownership of multiple records at once, you should use the ‘Batch Transfer Wizard’ instead of the above page. Click on the ‘Batch Transfer Wizard’ link in the admin homepage and follow the wizard instructions. The ‘Batch Transfer Wizard’ allows you also to move members of a group to another group if you want to.

Other features of the admin interface

- In addition to the above, you can use the admin interface to send email notifications to all groups by clicking the ‘Send a message to all groups’ link in the admin homepage.
- The ‘Admin settings’ link in the admin homepage allows you to adjust several administrative settings. These include:
 - Changing the admin username and password.
 - Changing the name and email used for the sender details when sending email notifications to groups or members.
 - Define/customize up to 4 info fields that new members are asked to fill during sign-up.
 - Customize the contents of the email sent to new members when they are approved.
 - Customize the date format used to display dates in the admin interface.
 - Customize the number of rows to display per page in the ‘View Groups’, ‘View Members’ and ‘View Member’s Records’ pages.
 - Define the default sign-up mode for new groups.
 - Change the name of the anonymous group and anonymous member.

Shortcut keys

The following shortcut keys can be used in AppGini apps as of AppGini 5.90.

Note: Unless otherwise specified (1) , pressing the shortcut keys below focuses (highlights) the related link/button (rather than performing the action itself -- this is intended to prevent doing any unintended action by mistake if pressing the wrong shortcut key combination). After a link is focused, you could press **ENTER** to actually activate the link, **TAB** to navigate to the next link, or **SHIFT + TAB** to navigate to the previous link.

(1) Wherever the shortcut listings below mention "*instant action*" , this is an exception to the above rule, meaning that pressing this particular shortcut keys combination would instantly apply the related action.

General shortcuts available in all pages

- **CTRL** or **SHIFT + F1** : Display available shortcuts (and enable/disabled shortcut keys).
- **ALT + 0** : Homepage.
- **ALT + 1 to 9** : Open available navigation menus.
- **ALT + M** : Import CSV data (if you have permission to).
- **ALT + A** : Admin area (if you're signed in as admin).
- **ALT + P** : User profile.
- **ALT + P** then **SHIFT + TAB** : Sign out.

Homepage

- **SHIFT + F2** : Highlight first table group.
- **CTRL + F2** : Highlight the first table link.

Table view

- **ALT + Q** : Quick search (type your search then press **ENTER**).
- **ALT + SHIFT + Q**: Clear quick search (*instant action*).
- **F2** : same as **ALT + Q**.
- **SHIFT + F2** : First button in the buttons bar above the table.
- **ALT + F2** : First element in the bottom navigation bar.
- **CTRL + F2** : First record selector checkbox.
 - You can then navigate to other records using **CTRL + ↑** and **CTRL + ↓**.
 - **Tip:** to open the currently highlighted record in the detail view, press **TAB** then **ENTER**.
- **ALT + CTRL + S** : Select/unselect all records.
- **ALT + CTRL + M** : Open **More** menu.
- **CTRL + ←** : Previous page (*instant action*).
- **CTRL + →** : Next page (*instant action*).

Detail view

- **F2** : First field in form.
- **SHIFT + F2** : First button in the action buttons at the right of the form.
- **CTRL + F2** : First child record.
- **ALT + I**: Show/hide admin information menu (if you're signed in as admin).
- **F8** : First child link (from the child links above the detail view form).
- **SHIFT + F8** : First child tab.
- **ALT + F8** : First navigation button in child tab.
- **CTRL + ←** : Previous record (*instant action*).
- **CTRL + →** : Next record (*instant action*).
- **CTRL + ↑** : Same as **CTRL + ←**.
- **CTRL + ↓** : Same as **CTRL + →**.
- **CTRL + ENTER** : Save Changes (*instant action*).
- **ALT + X** : Back to table view, discarding any changes made (*instant action*).

Filters page:

- **F2** : First filter.
- **SHIFT + F2** : Apply filters button.
- **CTRL + ENTER** : Apply filters (*instant action*).
- **CTRL + SHIFT + ENTER** : Save and apply filters (*instant action*).
- **ALT + X** : Cancel and go back to table view (*instant action*).

Print preview of table view:

- **SHIFT + F2** : Print button.
- **ALT + X** : Back (*instant action*).

Print preview of detail view:

- **SHIFT + F2** : Print button.
- **SHIFT + F8** : First child tab. You can then press **ENTER** to expand/collapse children records.
- **ALT + X** : Back to detail view (*instant action*).
- **CTRL + ←** : Previous record's detail view (*instant action*).
- **CTRL + →** : Next record's detail view (*instant action*).
- **CTRL + ↑** : Same as **CTRL + ←**.
- **CTRL + ↓** : Same as **CTRL + →**.

(AppGini 5.91+) Hide 'keyboard shortcuts reference' link

In AppGini 5.91 and above, the shortcuts window includes a link at the bottom that points to this page. This makes it easy for your app users to see a full reference. You can remove this link if desired by adding this line to `hooks/footer-extras.php`:

```
<script>_noShortcutsReference = true;</script>
```

LDAP Authentication

AppGini applications now support LDAP integration starting from version 24.10, providing a more streamlined login process for users who are already part of an LDAP directory. Here's how you can set up LDAP integration within your AppGini application.

Kindly note that LDAP authentication is available only in AppGini Pro.

Video overview of LDAP settings in AppGini apps

Your browser does not support the video tag.

Enabling LDAP Extension in PHP

Before you begin, ensure that the LDAP extension in PHP is enabled, as the integration is disabled by default. Upon enabling this extension, a new 'LDAP settings' tab will become available within the admin settings page of your AppGini application.

One way to check if LDAP extension is enabled is to sign in to your AppGini app as admin, go to the admin area > Utilities menu > Server status. In the server status page, under the PHP section, you should see LDAP details like the screenshot below if LDAP is enabled:

Idap

LDAP Support	enabled	
Total Links	0/unlimited	
API Version	3001	
Vendor Name	OpenLDAP	
Vendor Version	20513	
Directive	Local Value	Master Value
ldap.max_links	Unlimited	Unlimited

Figure 70: LDAP info section under PHP info

Configuring LDAP Settings

Sign in to your AppGini app as administrator. Go to the admin area, open the Utilities menu, and click on Admin settings. If the LDAP extension is enabled in PHP, you should see an 'LDAP settings' tab. Once you access the 'LDAP settings' tab, you can configure the LDAP integration. This section allows you to switch from the default login method, where AppGini manages usernames and passwords, to an LDAP-based authentication system.

Admin Area
Groups
Members
Utilities
Plugins
User's area
Sign Out

Admin Settings

Appearance
Sign up
Mail
Preconfigured users and groups
Application
LDAP settings

Before enabling LDAP login, make sure the admin username `admin` exists in the LDAP server. Otherwise, you will not be able to sign in as admin.

Login method
☐ Default
☒ LDAP

LDAP server

Examples: `ldap.example.com`, `ldaps://ldap.example.com`, `ldaps://ldap.example.com:636`, ...

LDAP version

Examples: `2`, `3`

LDAP username prefix

Examples: `FDC\`, `cn=`, `uid=`, ...

LDAP username suffix

Example: `,ou=people,dc=ldap,dc=example,dc=com`

Default user group for new LDAP users
☐ Disable (only admins can add users)
☒ Consultants
☐ Purchasing
☐ Sales

Figure 71: AppGini LDAP settings

Specifying the LDAP Server

Input your LDAP server URL in the format `ldap.example.com` or `ldaps://ldap.example.com` for SSL connections. If your LDAP server operates on a non-standard port, you can specify it like `ldap.example.com:389`.

LDAP Version

Select the LDAP protocol version that corresponds to your server's configuration. Most servers will work with version 3, which is recommended for optimal compatibility.

User DN (Distinguished Name) Pattern

You must specify the pattern for wrapping the username for LDAP logins. This pattern usually includes prefixes and suffixes, such as:

```
uid=USERNAME,ou=people,dc=ldap,dc=example,dc=com
```

In this example, the username prefix is `uid=`, and the username suffix is `,ou=people,dc=ldap,dc=example,dc=com` (note the initial comma `,`). Adjust the domain components (`dc=`) according to your LDAP server's domain.

Handling Non-Existent Users

Determine how AppGini should handle login attempts from users who are authenticated through LDAP but do not exist in the AppGini database. You can choose to:

- Disable login for such users, requiring an admin to manually add them, or
- Automatically create a user account in AppGini and assign the user to a default group that you can specify.

Note that the list of groups doesn't include the Admins group to prevent unintended privilege escalation of normal users. If you need to assign admin rights to an LDAP user, you must do so manually from the admin area.

Important Considerations

Before enabling LDAP authentication, confirm that your AppGini admin username exists in the LDAP directory. Otherwise, you'll be unable to log in to AppGini with admin privileges. If this does occur, you will need to edit the `config.php` file manually to revert to the default login method. This can be done by changing the line:

```
'loginMethod' => "ldap",
```

to:

```
'loginMethod' => "default",
```

Testing LDAP Integration

After saving the changes, it's wise to test the integration, without signing out from your admin account. To do so, open an anonymous (i.e. incognito or private) browser window, visit your AppGini application, and attempt to log in with an LDAP user. If the setup is correct, users should be able to sign in with their LDAP credentials and be automatically added to the specified default group if they don't already exist in AppGini.

If you're unable to sign in as an LDAP user, this could be due to incorrect LDAP settings. In that case, switch to the browser window where you are signed in as admin, try to adjust the settings, then switch back to the anonymous window and try to sign in.

Troubleshooting

If you're unable to sign in after enabling LDAP, and you get locked out of your admin account, you can manually disable LDAP. To do so, you will need to edit the `config.php` file to revert to the default login method. This can be done by changing the line:

```
'loginMethod' => "ldap",
```

to:

```
'loginMethod' => "default",
```

Conclusion

By following these steps, you can effectively integrate LDAP authentication into your AppGini application, leveraging existing user accounts and streamlining the login process. Make sure to thoroughly test the configuration with different user scenarios to ensure a smooth transition.

Enabling LDAP Extension in PHP

Before configuring LDAP settings in your AppGini app, you need to ensure that the LDAP extension in PHP is enabled. This process varies depending on your operating system.

For Linux (Debian/Ubuntu)

On Debian-based systems like Ubuntu, you can enable the LDAP extension for PHP by installing the required packages. Open a terminal and run the following commands:

```
sudo apt update
sudo apt install php-ldap
```

After installation, restart the Apache server to apply the changes:

```
sudo systemctl restart apache2
```

For Linux (CentOS/RHEL)

For CentOS or Red Hat-based distributions, use the following commands in the terminal:

```
sudo yum update
sudo yum install php-ldap
```

Then, restart the Apache server:

```
sudo systemctl restart httpd
```

For Windows

Enabling the LDAP extension in Windows is done through the `php.ini` file. Follow these steps:

1. Locate your `php.ini` file, which is usually found in your PHP installation directory, e.g., `C:\php\php.ini`.
2. Open `php.ini` in a text editor with administrative privileges.
3. Search for the line `;extension=ldap`. If the line starts with a semicolon (;), it's commented out.
4. Remove the semicolon to enable the extension. It should look like this:
`extension=ldap`
5. Save the `php.ini` file and restart your web server for the changes to take effect.

For WAMP, XAMPP, or other integrated server packages, you might be able to enable the LDAP extension through their respective control panels, usually by ticking a checkbox or switching a toggle next to the PHP LDAP extension.

Verifying LDAP Extension Activation

To confirm that the LDAP extension is active, create a PHP file with the following content and navigate to it in your web browser:

```
<?php phpinfo();
```

This outputs information about your PHP configuration. Look for a section titled 'ldap'. If it's present, the LDAP extension is enabled and working.

After enabling the LDAP extension in PHP, a new 'LDAP Settings' tab will become available within the admin settings page of your AppGini application. You can now proceed to configure the LDAP settings as described in the sections above.

Remember to keep your PHP environment secure and up to date, as enabling extensions can expose new vectors for potential vulnerabilities if not managed properly.

Advanced topics

- Hooks
 - The “hooks” folder
 - Global hooks
 - * login_failed() hook
 - * login_ok() hook
 - * member_activity() hook
 - Table-specific hooks
 - * tablename_before_insert() hook
 - * tablename_after_insert() hook
 - * tablename_before_update() hook
 - * tablename_after_update() hook
 - * tablename_before_delete() hook
 - * tablename_after_delete() hook
 - * tablename_dv() hook
 - * tablename_csv() hook
 - * tablename_init() hook
 - * tablename_header() hook
 - * tablename_footer() hook
 - * tablename_batch_actions() hook
 - DataList object
 - memberInfo array
 - Magic files
 - WindowMessages class
- Table and detail view classes
- Custom pages
- Third party libraries
- Command-line parameters
- URL parameters
- Troubleshooting errors and blank pages

Hooks (AKA events)

AppGini Hooks (events) are means of advanced customization of AppGini-generated apps. They allow you to customize your application behavior in a way that is separate from the generated code. This way, your custom code doesn't get overwritten if you regenerate your app later, and your project is ready for use directly after code generation without any further modifications.

Hooks work by intercepting users' actions (inserts, deletes, edits, selection of records, ... etc), and controlling what happens before and after these actions.

How do hooks work?

To use hooks, you should place your code modifications in the generated `hooks` folder. This folder contains a set of files that AppGini creates only once and they don't get overwritten later. These files contain hook functions that you can define. Your AppGini app calls these functions when performing specific tasks and executes the code you define in them.

For example, to send a notification email when a new order is added to the `orders` table, you should add the mail sending code in the `orders_after_insert()` function inside the `hooks/orders.php` file. This function is automatically called by the AppGini-generated application whenever a new record is created in the `orders` table. Any code you place inside that function is executed when a new record is added to that table through the AppGini-generated interface.

Global hooks

Global hook functions are defined in the generated `hooks/__global.php` file. This file contains hook functions that get called when a new member signs up, when a member signs in successfully and when a member fails to sign in. You could also define your own PHP functions here and they'll be visible to all your AppGini application pages.

The following hook functions are defined in this file:

- `login_ok()`
- `login_failed()`
- `member_activity()`
- `sendmail_handler()`
- `child_records_config()`

`login_ok()`

This hook function is called when a member successfully signs in. It can be used for example to redirect members to specific pages rather than the home page, or to save a log of members' activity, ... etc. If you open the generated `hooks/__global.php` file in a text editor, you can see this function defined as follows:

```
function login_ok($memberInfo, &$args) {  
  
    return '';  
}
```

Parameters:

- `$memberInfo` is an array containing details of the member who signed in. Please refer to `memberInfo` for more details.
- `$args` is currently not implemented but is reserved for future use.

Return value:

A string containing the URL to redirect the member to. It can be a relative or absolute URL. If the return string is empty, the member is redirected to the homepage (`index.php`), which is the default behavior.

Example:

Let's add code to save a log of members' login activity. Each time a member signs in, we'll record his username, IP address, login date and time into a log file. Here's how the hook function looks like after adding this code:

```
function login_ok($memberInfo, &$args) {  
    // the log file where we'll save member activity  
    $logFile = 'members.log';  
  
    // the member details we'll be saving into the file  
    $username = $memberInfo['username'];
```

```

$ip = $memberInfo['IP'];
$date = date('m/d/Y');
$time = date('h:i:s a');

// open the log file and append member login details
file_put_contents($logFile, "$date,$time,$username,$ip\n", FILE_APPEND);

return '';
}

```

login_failed()

This hook function is called when a login attempt fails. It can be used for example to log login errors. If you open the generated `hooks/__global.php` file in a text editor, you can see this function defined as follows:

```

function login_failed($attempt, &$amp;args) {

}

```

Parameters:

- `$attempt` is an associative array containing details of the failed login attempt. It contains the following keys:
 - `username`: the username entered during the login attempt.
 - `password`: the password entered during the login attempt.
 - `IP`: the IP address of the client attempting to log in.
- `$args` is currently not implemented but is reserved for future use.

Return value:

None.

Example:

To notify the admin when a user fails to log in, we can add this code into the `login_failed()` function:

```

function login_failed($attempt, &$amp;args){
    // email of admin
    $adminEmail = 'admin@domain.com';

    // someone trying to log as admin?
    if($attempt['username'] == 'admin'){

        // send the email
        @mail(
            $adminEmail, // email recipient
            "Failed login attempt", // email subject
            "Someone from {$attempt['IP']} tried to log in ".
            "as admin using the password {$attempt['password']}.", // message
            "From: $adminEmail"
        );
    }
}

```

member_activity()

This hook function is called when a new member signs up. If you open the generated `hooks/__global.php` file in a text editor, you can see this function defined as follows:

```
function member_activity($memberInfo, $activity, &$args){
    switch($activity){
        case 'pending':
            break;

        case 'automatic':
            break;

        case 'profile':
            break;

        case 'password':
            break;
    }
}
```

Parameters:

- `$memberInfo` is an associative array containing details of the member who signed up. Please refer to `memberInfo` for more details.
- `$activity` is a string indicating the type of activity. It can be one of the following values:
 - `pending`: the member signed up but his account is pending approval by the admin.
 - `automatic`: the member signed up and his account is automatically approved.
 - `profile`: the member updated his profile.
 - `password`: the member changed his password.
- `$args` is currently not implemented but is reserved for future use.

Return value:

None.

Example:

This example sends a welcome email to new users who were automatically approved, and a ‘please wait’ email for new users pending approval.

```
function member_activity($memberInfo, $activity, &$args){
    switch($activity){
        case 'pending':
            // send 'please wait' email to new user
            @mail(
                $memberInfo['email'], // email recipient
                "Thank you for signing up at our website!", // subject

                "Dear {$memberInfo['username']}, \n\n".
                "We'll review and approve your new account within a few hours.\n\n".
                "Thank you.", // message

                "From: support@domain.com" // the "From" address the user will see
            );
            break;

        case 'automatic':
            // send 'welcome' email to new user
            @mail(
                $memberInfo['email'], // email recipient
                "Thank you for signing up at our website!", // subject
            );
    }
}
```

```

        "Dear {$memberInfo['username']}, \n\n".
        "You can now log into our website from this page:\n".
        "http://www.domain.com/appgini\n\n".
        "Thank you.", // message

        "From: support@domain.com" // the "From" address the user will see
    );
    break;

    case 'profile':
        break;

    case 'password':
        break;

}
}

```

sendmail_handler()

This hook function is called when AppGini sends an email using the `sendmail()` function. It can be used to modify the email before it's sent. If you open the generated `hooks/__global.php` file in a text editor, you can see this function defined as follows:

```

function sendmail_handler(&$pm) {

}

```

Parameters:

- `$pm` is a PHPMailer object, passed by reference. Please refer to PHPMailer project on Github for more details.

Return value:

None.

child_records_config()

This hook function was added in AppGini 22.14, and can be used to modify the default configuration of the child records section in the detail view.

If you open the generated `hooks/__global.php` file in a text editor, you can see this function defined as follows:

```

function child_records_config($childTable, $childLookupField, &$config) {

}

```

Parameters:

- `$childTable` is the name of the child table.
- `$childLookupField` is the name of the lookup field in the child table.
- `$config` is an associative array containing the configuration for displaying child records for the current user, passed by reference. The default configuration, is stored in the `$pcConfig` array defined in the generated `parent-children.php` file.

Return value:

None.

Table-specific hooks

For each table in your project, AppGini generates a hook file named the same as the table name inside the **hooks** folder. This file contains hook functions that get called when a new record is added, when a record is edited, when a record is deleted, ... etc. These hooks are table-specific. That's why each table in your project has its own hook file.

The following hook functions are defined in this file:

- `tablename_before_insert()`
- `tablename_after_insert()`
- `tablename_before_update()`
- `tablename_after_update()`
- `tablename_before_delete()`
- `tablename_after_delete()`
- `tablename_dv()`
- `tablename_csv()`
- `tablename_init()`
- `tablename_header()`
- `tablename_footer()`
- `tablename_batch_actions()`

`tablename_init()`

Called before rendering the page. This is a very powerful hook that allows you to control all aspects of how the page is rendered. If you open the generated `hooks/tablename.php` file in a text editor (where `tablename` is the name of the concerned table), you can see this function defined as follows:

```
function tablename_init(&$options, $memberInfo, &$args) {  
  
    return true;  
}
```

Parameters

- `$options` (passed by reference so that it can be modified inside this hook function) a `DataList` object that sets options for rendering the page. Please refer to `DataList` for more details.
- `$memberInfo` is an array containing details of the member who signed in. Please refer to `memberInfo` for more details.
- `$args` is currently not used but is reserved for future uses.

Return value

`true` to render the page. `false` to cancel the operation (which could be useful for error handling to display an error message to the user and stop displaying any data).

Example

The following example checks that the logged user belongs to the admin group and accordingly allows CSV downloading of records. If the user is not a member of the admin group, CSV downloads are disabled.

```
function tablename_init(&$options, $memberInfo, &$args) {

    if($memberInfo['group'] == 'Admins') {
        $options->AllowCSV = 1;
    } else {
        $options->AllowCSV = 0;
    }

    return true;
}
```

There is another example in the Tips and tutorials section that uses the `tablename_init` hook to modify part of the table view query. Another example uses the `tablename_init` hook to apply a default filter to a table.

tablename_header()

Called before displaying page content. Can be used to return a customized header template for the table. If you open the generated `hooks/tablename.php` file in a text editor (where `tablename` is the name of the concerned table), you can see this function defined as follows:

```
function tablename_header($contentType, $memberInfo, &$args) {
    $header='';

    switch($contentType) {
        case 'tableview':
            $header='';
            break;

        case 'detailview':
            $header='';
            break;

        case 'tableview+detailview':
            $header='';
            break;

        case 'print-tableview':
            $header='';
            break;

        case 'print-detailview':
            $header='';
            break;

        case 'filters':
            $header='';
            break;
    }

    return $header;
}
```

Parameters

- `$contentType` specifies the type of view that will be displayed. Takes one of the following values: `tableview`, `detailview`, `tableview+detailview`, `print-tableview`, `print-detailview` or `filters`.

- `$memberInfo` is an array containing details of the member who signed in. Please refer to `memberInfo` for more details.
- `$args` is currently not used but is reserved for future uses.

Return value

String containing the HTML header code. If empty, the default `header.php` is used. If you want to include the default header besides your customized header, include the `<%%HEADER%%>` placeholder in the returned string. Note: If you have a customized `header-extras.php` file (see the contents of the `hooks` folder for more info), it won't be included in the page if you don't include the `<%%HEADER%%>` placeholder in the return string.

Example

The following example displays today's date and current time above the print-preview pages, so that the printed document shows this data. Notice that the placeholder `<%%HEADER%%>` is included so that the original header is still output to users. The modified code is at lines 18 and 22.

```
function tablename_header($contentType, $memberInfo, &$args) {
    $header='';

    switch($contentType) {
        case 'tableview':
            $header='';
            break;

        case 'detailview':
            $header='';
            break;

        case 'tableview+detailview':
            $header='';
            break;

        case 'print-tableview':
            $header='<%%HEADER%%><div align="right">'.date('r').'</div>';
            break;

        case 'print-detailview':
            $header='<%%HEADER%%><div align="right">'.date('r').'</div>';
            break;

        case 'filters':
            $header='';
            break;
    }

    return $header;
}
```

tablename_footer()

Called after displaying page content. Can be used to return a customized footer template for the table. If you open the generated `hooks/tablename.php` file in a text editor (where `tablename` is the name of the concerned table), you can see this function defined as follows:

```
function tablename_footer($contentType, $memberInfo, &$args) {
    $footer='';
```

```

switch($contentType) {
    case 'tableview':
        $footer='';
        break;

    case 'detailview':
        $footer='';
        break;

    case 'tableview+detailview':
        $footer='';
        break;

    case 'print-tableview':
        $footer='';
        break;

    case 'print-detailview':
        $footer='';
        break;

    case 'filters':
        $footer='';
        break;
}

return $footer;
}

```

Parameters

- `$contentType` specifies the type of view that will be displayed. Takes one of the following values: `tableview`, `detailview`, `tableview+detailview`, `print-tableview`, `print-detailview` or `filters`.
- `$memberInfo` is an array containing details of the member who signed in. Please refer to `memberInfo` for more details.
- `$args` is currently not used but is reserved for future uses.

Return value

String containing the HTML footer code. If empty, the default `footer.php` is used. If you want to include the default footer besides your customized footer, include the `<%%FOOTER%%>` placeholder in the returned string. Note: If you have a customized `footer-extras.php` file (see the contents of the `hooks` folder for more info), it won't be included in the page if you don't include the `<%%FOOTER%%>` placeholder in the return string.

Example

Please refer to the above example for `tablename_header`.

`tablename_before_insert()`

Called before executing the insert query. If you open the generated `hooks/tablename.php` file in a text editor (where `tablename` is the name of the concerned table), you can see this function defined as follows:

```

function tablename_before_insert(&$data, $memberInfo, &$args) {

    return true;
}

```

Parameters

- `$data` An associative array where the keys are field names and the values are the field data values to be inserted into the new record. This array is passed by reference so that modifications to it apply to the insert query.
- `$memberInfo` is an array containing details of the member who signed in. Please refer to `memberInfo` for more details.
- `$args` was not in use prior to AppGini 5.90. As of AppGini 5.90, it's used for exchanging further data as follows:
 - `$args['error_message']` can be set inside the hook function to display an error message to user in case of returning `false`.

Return value

A boolean `true` to perform the insert operation, or `false` to cancel it.

As of *AppGini 5.90*, if returning `false`, an error message string (no HTML tags allowed) can be displayed to users by passing it through `$args['error_message']`.

Example 1

In this example, let's assume that our table contains the fields: `unit_price`, `quantity` and `total`. We want to automatically calculate the value of the `total` field by multiplying `quantity` and `unit_price`.

```
function tablename_before_insert(&$data, $memberInfo, &$args) {  
  
    $data['total'] = $data['quantity'] * $data['unit_price'];  
  
    return true;  
}
```

See also: Using lookup fields in calculations.

Example 2 (AppGini 5.90+)

In this example, let's assume that we have a `job_orders` table, and we want to make sure the `duration` field value must be higher than 3. If not, we'll reject the record with an error message.

```
function job_orders_before_insert(&$data, $memberInfo, &$args) {  
  
    if($data['duration'] <= 3) {  
        $args['error_message'] = 'Error: Duration must be higher than 3.';  
        return false;  
    }  
  
    return true;  
}
```

tablename_after_insert()

Called after executing the insert query (but before executing the ownership insert query). If you open the generated `hooks/tablename.php` file in a text editor (where `tablename` is the name of the concerned table), you can see this function defined as follows:

```
function tablename_after_insert($data, $memberInfo, &$args) {  
  
    return true;  
}
```

Parameters

- `$data` is an associative array where the keys are field names and the values are the field data values that were inserted into the new record. It also includes the item `$data['selectedID']` which stores the value of the primary key for the new record.
- `$memberInfo` is an array containing details of the member who signed in. Please refer to `memberInfo` for more details.
- `$args` is currently not used but is reserved for future uses.

Return value

A boolean `true` to perform the ownership insert operation or `false` to cancel it. Warning: if a `false` is returned, the new record will have no ownership info.

Example 1

The following example sends a notification email to an employee when a user submits a new record. The email contains the record data.

```
function tablename_after_insert($data, $memberInfo, &$amp;args) {

    // to compose a message containing the submitted data,
    // we need to iterate through the $data array
    foreach($data as $field => $value) {
        $messageData .= "$field: $value \n";
    }

    sendmail([
        'to' => 'employee@company.com',
        'name' => 'Recipient Name',
        'subject' => 'A new record needs your attention',
        'message' => "The following new record was submitted by {$memberInfo['username']}: \n\n" . $
    ]);

    return true;
}
```

Example 2

The following example works with apps created by AppGini 23.17 or above. It uses the `WindowMessages` class to display a custom message to the user after a record is inserted. In this example, we're displaying a sample instruction for the user to follow after he's added a new record to the orders table reminding him to add order items.

```
function orders_after_insert($data, $memberInfo, &$amp;args) {

    WindowMessages::add('Next step: Add order items for this order!');

    return true;
}
```

tablename_before_update()

Called before executing the update query. If you open the generated `hooks/tablename.php` file in a text editor (where `tablename` is the name of the concerned table), you can see this function defined as follows:

```
function tablename_before_update(&$data, $memberInfo, &$amp;args) {

    return true;
}
```

Parameters

- **\$data** An associative array where the keys are field names and the values are the new data values to update the field with. This array is passed by reference so that modifications to it apply to the update query. This array includes the item **\$data['selectedID']** which stores the value of the primary key for the record to be updated.
- **\$memberInfo** is an array containing details of the member who signed in. Please refer to memberInfo for more details.
- **\$args** was not in use prior to AppGini 5.90. As of AppGini 5.90, it's used for exchanging further data as follows:
 - **\$args['error_message']** can be set inside the hook function to display an error message to user in case of returning **false**.
 - **\$args['old_data']** is an associative array containing existing record values. This is useful for comparing the new values passed through the **\$data** parameter to the stored values in the record before the actual update operation is performed.

Return value

true to perform the update operation or **false** to cancel it.

As of AppGini 5.90, if returning **false**, an error message string (no HTML tags allowed) can be displayed to users by passing it through **\$args['error_message']** (See example 2 for `tablename__before__insert`).

Example

Let's say we have an orders table. When a user makes changes to a record and saves them, we want to automatically calculate the value of the *total* field using the fields *subtotal*, *discount* and *sales_tax*, where discount and sales_tax are stored as percentages (i.e. a discount value of 10 means 10% of subtotal):

```
function tablename_before_update(&$data, $memberInfo, &$args) {  
  
    // calculate total after applying discount  
    $data['total'] = $data['subtotal'] * (1 - $data['discount'] / 100);  
  
    // calculate total after applying sales tax  
    $data['total'] = $data['total'] * (1 + $data['sales_tax'] / 100);  
  
    return true;  
}
```

Another example

Let's say that we want to prevent updates to any records in a particular table that are older than 30 days. To do so, we would customize the `tablename__before__update()` hooks like this:

```
function tablename_before_update(&$data, $memberInfo, &$args) {  
  
    // get the creation date of the record  
    $creationDate=sqlValue("select dateAdded from membership_userrecords  
        where tableName='tablename' and pkValue='{ $data['selectedID'] }'");  
  
    // if the record is older than 30 days, deny changes  
    if($creationDate < strtotime('30 days ago')) return false;  
  
    return true;  
}
```

Don't forget to replace *tablename* at line 5 above, with the actual name of your table.

tablename_after_update()

Called after executing the update query and before executing the ownership update query. If you open the generated `hooks/tablename.php` file in a text editor (where `tablename` is the name of the concerned table), you can see this function defined as follows:

```
function tablename_after_update($data, $memberInfo, &$amp;args) {  
  
    return true;  
}
```

Parameters

- `$data` is an associative array where the keys are field names and the values are the field data values that were inserted into the new record. It also includes the item `$data['selectedID']` which stores the value of the primary key for the new record.
- `$memberInfo` is an array containing details of the member who signed in. Please refer to `memberInfo` for more details.
- `$args` was not in use prior to AppGini 5.90. As of AppGini 5.90, it's used for exchanging further data as follows:
 - `$args['old_data']` is an associative array containing old record values that existed before the update operation. This is useful for comparing the new values passed through the `$data` parameter to the old values of the record that existed before the update operation. You could use this for example for auditing purposes.

Return value

`true` to perform the ownership update operation or `false` to cancel it.

Example

Please refer to the example for `tablename_after_insert` hook above.

tablename_before_delete()

Called before deleting a record (and before performing child records check). If you open the generated `hooks/tablename.php` file in a text editor (where `tablename` is the name of the concerned table), you can see this function defined as follows:

```
function tablename_before_delete($selectedID, &$amp;$skipChecks, $memberInfo, &$amp;args) {  
  
    return true;  
}
```

Parameters

- `$selectedID` is the primary key value of the record to be deleted.
- `$skipChecks` is a flag passed by reference that determines whether child records check should be performed or not. If you set `$skipChecks` to `true` inside this hook function, no child records check will be made. If you set it to `false`, the check will be performed.
- `$memberInfo` is an array containing details of the member who signed in. Please refer to `memberInfo` for more details.
- `$args` is currently not used but is reserved for future uses.

Return value

`true` to perform the delete operation or `false` to cancel it.

Example

In this example, we'll assume that our table contains a checkbox field named *approved* . We want to allow deleting of the record only if that field is not checked (set to 0). If the field is checked (set to 1), it won't be deleted unless the user is a member of the Admins group.

```
function tablename_before_delete($selectedID, &$skipChecks, $memberInfo, &$args) {

    // We'll perform the 'approved' check only if the user
    // is not a member of the 'Admins' group.

    if($memberInfo['group'] != 'Admins') {
        $id=makeSafe($SelectedID);
        $approved=sqlValue("select `approved` from `tablename` where `id`='$id'");

        // if the record is approved, don't allow deleting it
        if($approved) return false;
    }

    return true;
}
```

We assumed in the above example that the primary key field of the table is named *id*. Also, notice in line 7 the use of the `makeSafe()` function, which prepares variables to be used safely inside SQL queries. In line 8, we used the `sqlValue()` function which performs a SQL query that we know returns a single value. It's a shortcut function that saves us the effort of processing a MySQL result set.

tablename_after_delete()

Called after deleting a record. If you open the generated `hooks/tablename.php` file in a text editor (where `tablename` is the name of the concerned table), you can see this function defined as follows:

```
function tablename_after_delete($selectedID, $memberInfo, &$args) {

}
```

Parameters

- `$selectedID` is the primary key value of the deleted record.
- `$memberInfo` is an array containing details of the member who signed in. Please refer to `memberInfo` for more details.
- `$args` is currently not used but is reserved for future uses.

Return value

None.

Example

This example logs the date and time a record was deleted and who deleted it.

```
function tablename_after_delete($selectedID, $memberInfo, &$args) {
    // log file
    $logFile='deletes.log';

    // attempt to open the log file for appending
    if(!$fp = @fopen($logFile, 'a')) return;

    // write log data: date/time, username, IP, record ID
    $datetime=date('r');
```

```

        fwrite($fp, "$datetime,{ $memberInfo['username'] },{ $memberInfo['IP'] },$selectedID\n");
        fclose($fp);
    }

```

tablename_dv()

Called when a user requests to view the detail view (before displaying the detail view). If you open the generated `hooks/tablename.php` file in a text editor (where `tablename` is the name of the concerned table), you can see this function defined as follows:

```

function tablename_dv($selectedID, $memberInfo, &$html, &$args) {

}

```

Parameters

- `$selectedID` The primary key value of the record selected. It's set to `false` if no record is selected (i.e. the detail view will be displayed to enter a new record).
- `$memberInfo` is an array containing details of the member who signed in. Please refer to `memberInfo` for more details.
- `$html` (passed by reference so that it can be modified inside this hook function) the HTML code of the form ready to be displayed. This could be useful for manipulating the code before displaying it using regular expressions, ... etc.
- `$args` is currently not used but is reserved for future uses.

Return value

None.

Example

The following example sets the price field as read-only for non-admin users. The example demonstrates how to "inject" JavaScript code to the detail view to change its behavior. Please note that setting a field as read-only via JavaScript is not sufficient to prevent modifying it. Power users can easily circumvent this. So, you have to also force this server-side, for example using the `before_update` hook .

```

function tablename_dv($selectedID, $memberInfo, &$html, &$args) {
    /* current user is not an admin? */
    if($mi['group'] != 'Admins') {
        ob_start();
        ?>
        <script>
            $j(function() {
                $j('#price').prop('readonly', true);
            })
        </script>
        <?php
            $html .= ob_get_clean();
        }
    }
}

```

tablename_csv()

Called when a user requests to download table data as a CSV file (by clicking the SAVE CSV button). If you open the generated `hooks/tablename.php` file in a text editor (where `tablename` is the name of the concerned table), you can see this function defined as follows:

```

function tablename_csv($query, $memberInfo, $args) {

```

```
    return $query;
}
```

Parameters

- `$query` contains the query that will be executed to return the data in the CSV file.
- `$memberInfo` is an array containing details of the member who signed in. Please refer to `memberInfo` for more details.
- `$args` is currently not used but is reserved for future uses.

Return value

A string containing the query to use for fetching the CSV data. If `false` or empty is returned, the default query is used.

Example

The following example modifies the SQL query used to limit records retrieved to 10 records only if the user requesting the CSV file is not an admin.

```
function tablename_csv($query, $memberInfo, $args) {

    // return only the first 10 records for non-admin users.
    if($memberInfo['group'] != 'Admins') {
        $query.=" limit 10";
    }

    return $query;
}
```

Using lookup fields in calculations

When customizing the `before_insert`, `after_insert`, `before_update` or `after_update` hooks to make a calculation, you might encounter a case where one or more fields in the formula you're calculating is a lookup field (foreign key) . In this case, the value of `$data['fieldname']` (where `fieldname` is the name of the concerned lookup field) is probably NOT the value you'd like to use for your calculation.

To explain that, let's have a brief look at how lookup fields work. A lookup field is a way of referencing a value from one table in another table. For example, we might be storing product unit price in the `products` table and want to use that same unit price in the `order_items` table without having to manually re-type the price ... this is important not just to save a few keystrokes during data entry, but to also ensure referential integrity ... If you throw the same product price into every table in your database, it will be a nightmare to update the price later and make sure all tables see the updated price.

To avoid that mess, we use lookup fields. A unit price lookup field in the `order_items` table doesn't store the actual price value but rather a reference value that points to the location of the unit price in the `products` table. The best possible reference to use is the primary key of the product. Let's have an example with numbers to see this in action.

Products table

ID	Product	Unit price
15	Lindt HELLO Crunchy Nougat	2.05
16	Lindt CREATION Crème Brûlée	2.35
18	Lindt EXCELLENCE Mint	3.25
19	Lindt CREATION Pistachio	3.25

That was yummy! Each entry in the above table has a primary key ID value, which doesn't tell much about the item itself but is used as a reference to it. So, if we talk about product #18, we know we are referring to *Lindt EXCELLENCE Mint* priced at \$3.25. Primary key fields are usually (but not necessarily) named ID.

Let's now have a look at some data from the `order_items` table.

Order Items table

ID	Order ID	Product	Unit price	Quantity	Subtotal
2024	305	15	15	1	
2025	305	18	18	3	
2026	306	18	18	1	
2027	307	19	19	2	

Similar to the `products` table, the ID column above is the primary key field of the `order_items` table, a way of uniquely identifying each row. `OrderID` is a lookup field to the orders table (not shown here as it's irrelevant to our discussion). `Product` and 'Unit Price' are both lookup fields to the products table. To understand this with an example, order item #2024 is an order for product #15, which is *Lindt HELLO*

Crunchy Nougat and its price is of course that of product #15 which is \$2.05. And the quantity of *Lindt HELLO Crunchy Nougat* ordered in this record is 1.

When your AppGini application displays the `order_items` table, it doesn't display reference values like the above. It automatically *joins* both tables and displays more human-readable results like the ones below

Order Items table joined with Products table

ID	Order ID	Product	Unit price	Quantity	Subtotal
2024	305	Lindt HELLO Crunchy Nougat	2.05	1	
2025	305	Lindt EXCELLENCE Mint	3.25	3	
2026	306	Lindt EXCELLENCE Mint	3.25	1	
2027	307	Lindt CREATION Pistachio	3.25	2	

If we later make any modifications to any product in the `products` table, like changing its name or unit price, the changes are automatically reflected in the `order_items` table without having to perform any manual data entry.

What remains now is to write code for calculating the subtotal column of the `order_items` table. We want this calculation to be applied whenever we add a new order item and also whenever we make changes to any existing order item. Therefore, we should perform the calculation in both the `before_insert` and `before_update` hook functions.

The initial code I see many AppGini users write usually looks something like this:

```
$data['Subtotal'] = $data['UnitPrice'] * $data['Quantity'];
```

The problem with the above code is that `$data['UnitPrice']` stores the primary key of the parent product (the value of the `ID` field from the parent record in `products`). For example, if we're calculating the subtotal of order item #2025, the above code would display a subtotal of $18 \times 3 = \$54$. This is of course not correct, as the unit price for *Lindt EXCELLENCE Mint* is \$3.25 and we have a quantity of 3 units. Therefore, the correct subtotal should be $3.25 \times 3 = \$9.75$.

What's wrong with the above code is that we didn't take into consideration the fact that `UnitPrice` field in `order_items` is actually a lookup field. The stored value is not the unit price but rather the primary key value of the parent product. Accordingly, we should retrieve the actual unit price from the `products` table using this code:

```
$UnitPrice = sqlValue(
    "SELECT UnitPrice FROM products where ID='{$data['UnitPrice']}'"
);
```

The above code retrieves the unit price from the `products` table given the primary key value stored in the child `order_items` table, `$data['UnitPrice']`, and stores the actual unit price in `$UnitPrice`. We can now perform the calculation as follows:

```
$data['Subtotal'] = $UnitPrice * $data['Quantity'];
```

Putting it all together, whenever we are performing calculations that involve lookup fields, we should first retrieve the actual values from the parent table and use those retrieved values in the calculation formula. It's very easy to write once we understand how it works. To sum up, here is our subtotal code:

```
$UnitPrice = sqlValue(
    "SELECT UnitPrice FROM products where ID='{$data['UnitPrice']}'"
);
$data['Subtotal'] = $UnitPrice * $data['Quantity'];
```

One final note ... some tables contain non-numeric primary key values. For example, if the above `products` table stores primary keys as `LHCN01`, `LEM01` ... etc rather than 18, 19 and so on, then we should escape those primary keys first to avoid query errors and protect against SQL injection attacks:

```

/* Escape non-numeric lookup values before using them in SQL queries */
$SafeUnitPriceLookup = makeSafe($data['UnitPrice']);

/*
    Now it's safe to use $SafeUnitPriceLookup to
    retrieve our unit price
*/
$UnitPrice = sqlValue(
    "SELECT UnitPrice FROM products where ID='{ $SafeUnitPriceLookup }'"
);

/* And here is our calculation */
$data['Subtotal'] = $UnitPrice * $data['Quantity'];

```

To summarize, whenever you are working with lookup fields in your calculations, you should first retrieve the actual values from the parent table and then use those values in your calculations. This is a simple concept that can save you a lot of time and headaches in the future.

Adding custom “batch actions” that apply to multiple records

When you select one or more records in the table view, a “More” button is displayed above the table. If you click that button, it opens the batch actions menu. This menu displays some actions that you can perform on the records you selected – see the screenshot below. Which actions show up in the menu depends on the permissions you have.

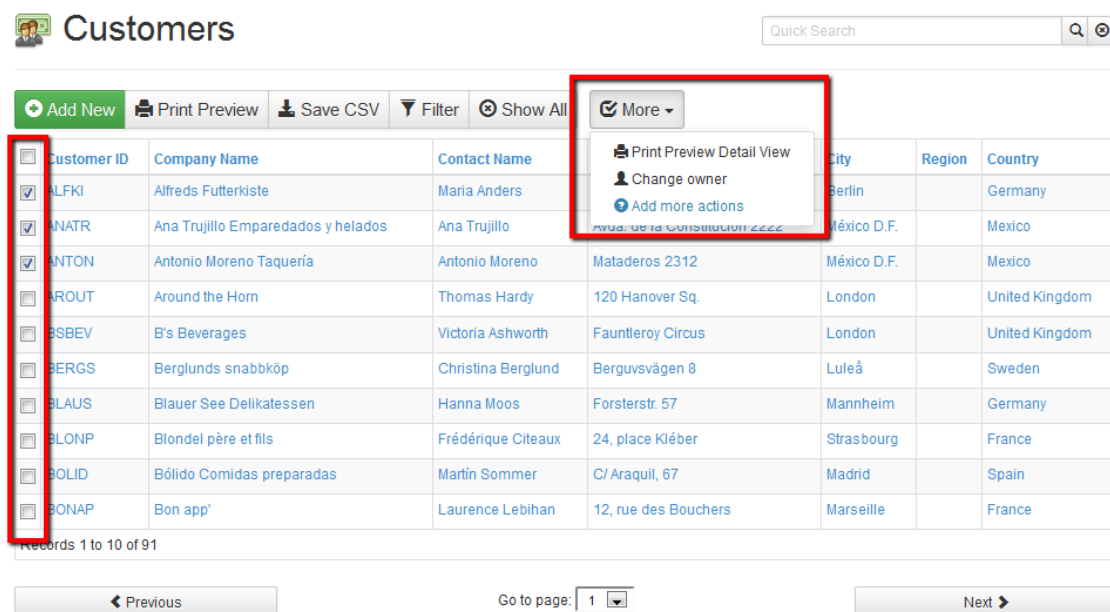


Figure 72: Batch actions menu

For example, if you are an admin, you can change the owner of the records. If you have delete permissions, and you’ve enabled mass-delete in AppGini, you can delete the records.

Adding custom batch actions

TIP!

Don’t have the time or programming knowledge to write your own batch actions? We have a plugin for that now! Check our Mass Update plugin. This plugin allows you to add as many batch actions as you want in a very short time, without writing a single line of code.

You can define your own batch actions inside the body of the `tablename_batch_actions()` function in the generated `hooks/tablename.php` file. In this function, you just define the name of the batch action. You can add the details and functionality of the batch action in another place that we’ll come to in a

moment. The `tablename_batch_actions()` hook works by returning an array of actions. Your AppGini application receives this array and displays the actions in the “More” menu.

When a user chooses an action from the “More” menu, your AppGini application calls the javascript function linked to that action. The name of this javascript function is part of the data in the array we mentioned above (the array returned from the `tablename_batch_actions` hook).

You should define the javascript function in the file `tablename-tv.js` inside the `hooks` folder. This function could do anything you want to apply to the selected records. It could open a new page, or make an ajax request, or any other action you wish to do. There is no specific implementation that you have to follow here. We’ll discuss an example action with all these details below so you can use it as a guideline.

This diagram explains how this all works.

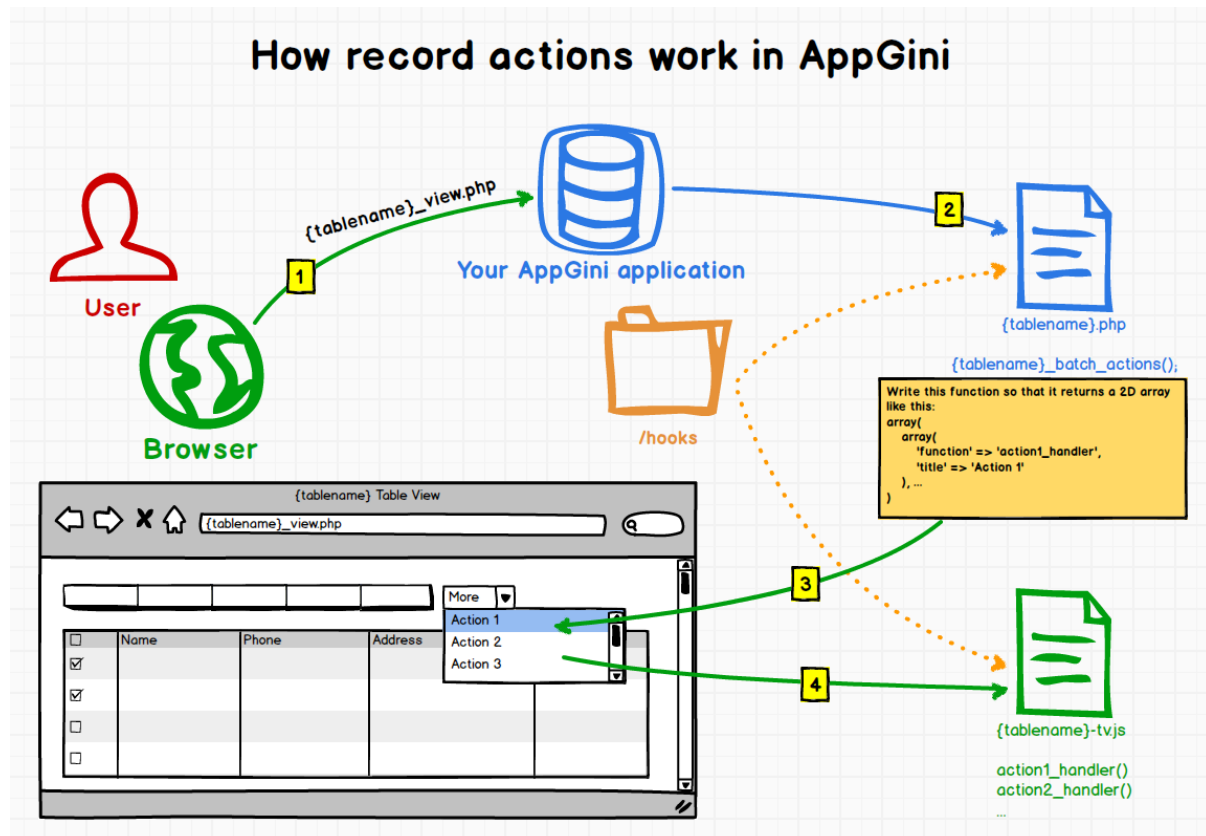


Figure 73: Batch actions diagram

So, here is the sequence of events:

1. The user opens the table view of a table in your AppGini application.
2. The application calls the hook function `tablename_batch_actions()`. This is where you define the extra actions users can choose.
3. This function returns an array that describes one or more actions and the name of the javascript function to call if the user selects an action. The application adds those actions to the “More” menu.
4. If the user selects one or more records, opens the “More” menu, and chooses one of the actions you defined in the `tablename_batch_actions()` hook, the application passes the IDs of the selected records to the javascript function you associated with that action.

Example: Adding a batch action to print mailing labels for selected records

Back to the customers table example:

Customer ID	Company Name	Contact Name	City	Region	Country
<input checked="" type="checkbox"/> ALFKI	Alfreds Futterkiste	Maria Anders	Berlin		Germany
<input checked="" type="checkbox"/> ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	México D.F.		Mexico
<input checked="" type="checkbox"/> ANTON	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	Mexico
<input type="checkbox"/> AROUT	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	United Kingdom
<input type="checkbox"/> BSBEV	B's Beverages	Victoria Ashworth	Fauntleroy Circus	London	United Kingdom
<input type="checkbox"/> BERGS	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	Sweden
<input type="checkbox"/> BLAUS	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	Germany
<input type="checkbox"/> BLONP	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	France
<input type="checkbox"/> BOLID	Bólido Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid	Spain
<input type="checkbox"/> BONAP	Bon app'	Laurence Lebihan	12, rue des Bouchers	Marseille	France

Figure 74: Customers table

Let's say you want to add a batch action to print mailing labels for the selected customers. Here is how you can do it: the first step is to add the action into the `customers_batch_actions()` hook. To do so, we'll open the `hooks/customers.php` file, we should find our hook function:

```
function customers_batch_actions(&$args){

    return array();

}
```

The function above is empty (we call this a skeleton function). We need to add our action to it. So, let's modify it to read:

```
function customers_batch_actions(&$args){

    return [

        [

            'title' => 'Print mail labels',
            'function' => 'print_mail_labels',
            'icon' => 'th-list'

        ]

    ];

}
```

The code above tells our application to display an extra action in the “More” menu labeled “Print mail labels”. If a user chooses that action, the application will pass the IDs (primary key values) of the selected records to a javascript function named `print_mail_labels()`. We didn't write this function yet. We'll do so in a moment. But before we do so, let's take a look on the “More” menu after adding the code above.

We've specified an icon name in the code above. So, the icon shows up to the left of the new action. For a full list of supported icon names, please refer to the Bootstrap Glyphicons list. All icons there have a name like “glyphicon-xyz” ... just use the xyz part in our hook code to specify an icon.

TIP!

To display the batch action only to users from a specific group, you can add a conditional

Customers

+ Add New Print Preview Save CSV Filter Show All More ▾			
<input type="checkbox"/> Customer ID	Company Name		
<input type="checkbox"/> ALFKI	Alfreds Futterkiste		
<input type="checkbox"/> ANATR	Ana Trujillo Emparedados y helados		
<input type="checkbox"/> ANTON	Antonio Moreno Taquería	Antonio Moreno	
<input type="checkbox"/> AROUT	Around the Horn	Hardy	121
<input checked="" type="checkbox"/> BSBEV	B's Beverages	Shworth	Fai
<input checked="" type="checkbox"/> BERGS	Berglunds snabbköp	Berglund	Be
<input type="checkbox"/> BLAUS	Blauer See Delikatessen	Hanna Moos	Foi
<input type="checkbox"/> BLONP	Blondel père et fils	Frédérique Citeaux	24,

[Print Preview Detail View](#)
[Change owner](#)
[Add more actions](#)
[Print mail labels](#)

This item was added using the batch_actions hook.

Figure 75: More menu with print mail labels action

check in the hook function. For example, to display the action only to users in the ‘Admins’ group, you can add the following code:

```
$memberInfo = getMemberInfo();
// if the current user is not an admin, return an empty array
if($memberInfo['group'] != 'Admins') return [];

return [ ... ]; // your batch actions array here
```

The next step is to define the `print_mail_labels()` javascript function. This is the function that our application would call if the user clicks the “Print mail labels” item in the menu. We should write this function in the `customers-tv.js` file in the `hooks` folder ... If you don’t find that file in the folder, just create it there .. the format is `tablename-tv.js` (where `tablename` is the name of the concerned table). If the file exists in the `hooks` folder, it’s loaded in the table view. So, whatever javascript code you put there will get executed in the table view of the concerned table.

Let’s write our code in the `customers-tv.js` file as follows:

```
function print_mail_labels(table_name, ids) {
    alert("IDs selected from " + table_name + ": " + ids);
}
```

Here is what happens when we choose the “Print mail labels” action after adding the above code:

The above code simply displays the parameters passed to the `print_mail_labels()` function. When you write the javascript function, you should write it so that it receives two parameters. The first one is a string containing the table name (this is useful if you have one function for handling multiple tables), and the second one is an array of selected record IDs (primary key values of selected records).

Let’s change the javascript code to do something more useful. We’ll pass the selected IDs to a PHP script to display the mail labels for those records. So let’s rewrite the `print_mail_labels()` function as follows.

```
function print_mail_labels(table_name, ids) {
    /*
        we'll open the mail labels page in a new window
        that page is a server-side PHP script named mail-labels.php
        but first, let's prepare the parameters to send to that script
    */
```

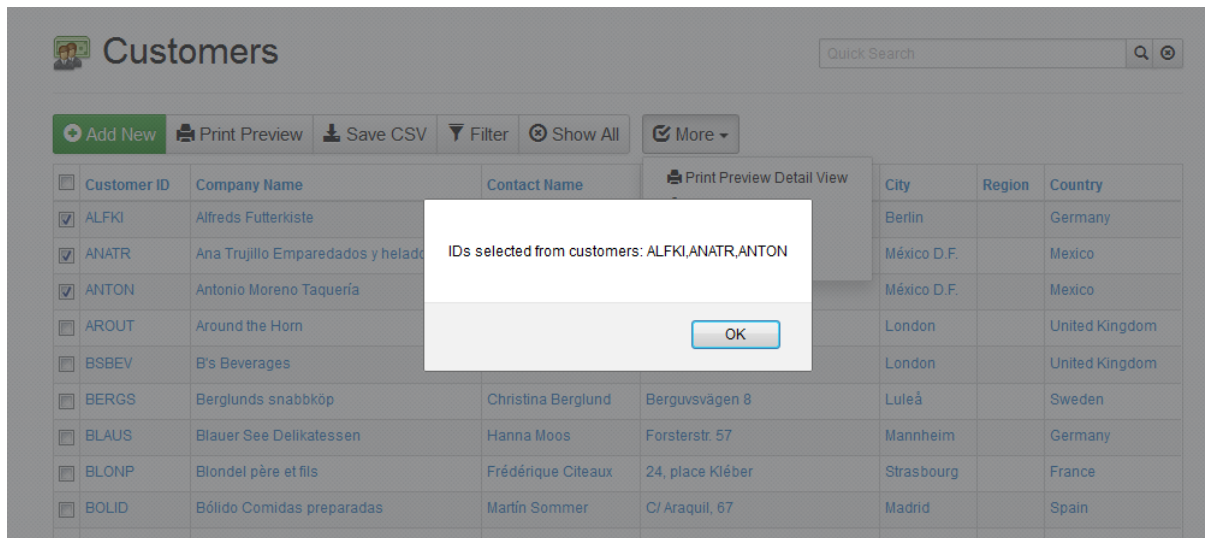


Figure 76: Alert showing selected IDs

```
var url = 'mail-labels.php?table=' + table_name;
for(var i = 0; i < ids.length; i++){
    url = url + '&'
        + encodeURIComponent('ids[]') + '='
        + encodeURIComponent(ids[i]);
}

window.open(url);
}
```

Finally, let's write the server-side `mail-labels.php` script. Based on the code above, we assumed the location of this script to be the main folder of our AppGini application. Here is how this script might look like:

```
<?php
/*
    Including the following files allows us to use many shortcut
    functions provided by AppGini. Here, we'll be using the
    following functions:
    makeSafe()
        protect against malicious SQL injection attacks
    sql()
        connect to the database and execute a SQL query
    db_fetch_assoc()
        same as PHP built-in mysqli_fetch_assoc() function
*/
include(__DIR__ . "/lib.php");

/* receive calling parameters */
$table = $_REQUEST['table'];
$ids = $_REQUEST['ids']; /* this is an array of IDs */

/* a comma-separated list of IDs to use in the query */
$cs_ids = '';
foreach($ids as $id){
    $cs_ids .= "'" . makeSafe($id) . "',";
}
$cs_ids = substr($cs_ids, 0, -1); /* remove last comma */
```

```

/* retrieve the records and display mail labels */
$eo = ['silentErrors' => true];
$res = sql("select * from customers " .
           "where CustomerID in ({ $cs_ids})", $eo);
while($row = db_fetch_assoc($res)){
    ?>
    <b><?php echo $row['CompanyName']; ?></b><br>
    <i>C/O <?php echo $row['ContactName']; ?></i><br>
    <?php echo $row['Address']; ?><br>
    <?php echo $row['City']; ?><br>
    <?php echo $row['Region']; ?>
    <?php echo $row['PostalCode']; ?><br>
    <?php echo $row['Country']; ?><br>
    <br>
    <br>
    <hr>
    <?php
}

```

Here is a sample of the output from the above script.

Alfreds Futterkiste

C/O Maria Anders

Obere Str. 57

Berlin

12209

Germany

Ana Trujillo Emparedados y helados

C/O Ana Trujillo

Avda. de la Constitución 2222

México D.F.

05021

Mexico

Antonio Moreno Taquería

C/O Antonio Moreno

Mataderos 2312

México D.F.

05023

Mexico

Berglunds snabbköp

C/O Christina Berglund

Berguvägen 8

Figure 77: Sample mail labels

We chose to implement the action handling using a javascript function to allow a lot of flexibility for customizations. In the above example, we prepared some parameters and opened a new page. You might instead wish to do something in the background by using an Ajax request without opening a new page. It's all up to you.

Note: The above example used the Northwind project, which is the same one used for our online demo. You can download the Northwind project file, application files and the sample data to experiment on your own.

TIP!

Don't have the time or programming knowledge to write your own batch actions? We have a plugin for that now! Check our Mass Update plugin. This plugin allows you to add as many batch actions as you want in a very short time, without writing a single line of code.

DataList object

The **DataList** object exposes many options that you can control to affect the behavior and appearance of each of the AppGini-generated table pages that users see.

DataList object is passed to the `tablename_init` hook function. This hook function is called before displaying data to users. So, you can control the various appearance and behavior options by modifying this object inside that hook function.

Here is a list of the editable properties of the DataList object

AllowCSV

Setting this property to 1 allows users to download table records as a CSV file. Setting it to 0 disables this.

AllowDeleteOfParents

Setting this property to 1 allows users who have delete permissions to delete a record even if it has child records in other tables. Setting it to 0 disables this.

AllowFilters

Setting this property to 1 allows users to access the filters page to view and modify filters. Setting it to 0 disables this.

AllowPrinting

Setting this property to 1 allows users to access the ‘Print preview’ page. Setting it to 0 disables this.

AllowSavingFilters

Setting this property to 1 allows users to save filters as HTML code to access them quickly later. Setting it to 0 disables this.

AllowSorting

Setting this property to 1 allows users to sort table records. Setting it to 0 disables this.

CSVSeparator

Specifies the field separator to use when downloading data as a CSV file. The default is comma (,).

ColCaption

An array that specifies the titles of columns displayed in the table view.

ColNumber

An array that specifies which fields to use in the table view. It works by selecting some (or all) of the fields listed in the **QueryFieldsTV** property explained below.

ColWidth

An array that specifies the width of each column in the table view. If the **ShowTableHeader** property (explained below) is set to 1, the **ColWidth** property is overridden by the width values specified in the table view template file (`templates/tablename_templateTV.html`).

DefaultSortDirection

A string that can be set to 'asc' or 'desc'. Please see the **DefaultSortField** property below.

DefaultSortField

Specifies the field to use for default sorting of the table view records. This property can be set to a number to specify which field to sort by from the **QueryFieldsTV** property explained below. Alternatively, it can be set to a string specifying an explicit field name or MySQL expression to use for default sorting.

DVClasses

Was added in AppGini 5.60. Additional CSS classes to apply to the detail view container (space-separated)

FilterPage

Specifies a custom search page to use when users click on the **FILTERS** button. If no value is provided, the default filters page is used. You can use this feature to create advanced search forms for your tables. Please see [Creating customized search forms](#) for a detailed example.

PrimaryKey

A string that specifies the name of the primary key field for the table. You shouldn't change this value.

QueryFieldsCSV

An associative array specifying the fields used in the query that fetches data when users request to download a CSV file. The array keys represent the field names or MySQL expressions used in the query. The array values represent the column titles to display in the CSV file.

QueryFieldsFilters

An associative array specifying the fields used in the filters page. The array keys represent the field names. The array values represent the field titles to display in the filters page.

QueryFieldsTV

An associative array specifying the fields that can be displayed in the table view. The array keys represent the field names or MySQL expressions used in the query. The array values represent the column titles. The fields actually displayed in the table view are specified in the **ColNumber** array explained above.

QueryFrom

A string that specifies the contents of the FROM part of the query used in the table view and the CSV file.

QuickSearch

A number that specifies how to display the quick search box. It can take any of the following values:

- 0: no quick search box shown.
- 1: quick box shown on the top left of the table view.
- 2: quick box shown on the top center of the table view.
- 3: quick box shown on the top right of the table view.

Update: As of AppGini 5.20 and above, setting this property to 0 hides the quick search box, and setting it to any non-zero value displays the quick search box. The position of the box is determined by the screen size.

QuickSearchText

A string that specifies the title to display besides the quick search box.

RecordsPerPage

A number that specifies how many records to show per page in the table view.

RedirectAfterInsert

If users are allowed to add new records to the table, this property specifies the URL to which users will be redirected after adding the new record.

SelectedTemplate

A string that specifies the path to the HTML template file to use for formatting a currently-selected record in the table view.

SeparateDV

A number that is set to 1 to display the detail view in a separate page, or 0 to display it below the table view.

ShowTableHeader

A number that is set to 1 (the default) to display column titles above the table view. Table titles are specified in the `ColCaption` property explained above. If set to 0, column titles are not displayed (this is useful if you need to change the horizontal layout of fields in the template file `templates/tablename_templateTV.html` to a different non-horizontal layout).

TableTitle

The title that will be displayed above the table view.

Template

A string that specifies the path to the HTML template file to use for formatting all records in the table view except the currently-selected one.

TemplateDV

Was added in AppGini 5.61. A string that specifies the path (relative to the main directory of the application) to the HTML template file to use for displaying the detail view.

TemplateDVP

Was added in AppGini 5.61. A string that specifies the path (relative to the main directory of the application) to the HTML template file to use for displaying the print preview of the detail view.

TVClasses

Was added in AppGini 5.60. Additional CSS classes to apply to the table view container (space-separated)

Inspecting the DataList object

For debugging purposes, you can inspect the contents of the **DataList** object by adding the following code into the `tablename_init` hook function in the generated `hooks/tablename.php` file (replace `tablename` by the actual name of the concerned table):

```
function tablename_init(&$options, $memberInfo, &$args) {
    ob_start();
    $xc = get_object_vars($options);
    ksort($xc);
    print_r($xc);
    $c = ob_get_clean();
    echo "<pre>" . htmlspecialchars($c) . "</pre>";

    return TRUE;
}
```

The above code will output the contents of the **DataList** object to the browser above the table view. You can use this to inspect, debug and change the various properties. But you should use this carefully in a protected environment for testing purposes only.

memberInfo array

`$memberInfo` is an associative array containing logged member's info. The array contains the following keys:

- **username**: the member username.
- **groupID**: the numeric ID of the member's group.
- **group**: the name of the member's group.
- **admin**: true for admin member, false for others.
- **email**: the email address of the member.
- **IP**: the IP address from where the member is currently logged.
- **custom**: a numeric array containing the values of custom fields for the member. Custom fields can be defined via the admin settings page in the admin area of your AppGini application. Currently up to 4 custom fields are supported. So, to access the value of the first custom field for the member, you can use `$memberInfo['custom'][0]`.

The `$memberInfo` array is passed to many hook functions, both global and table-specific. For example, you can access the username of the currently logged member in a hook function by using `$memberInfo['username']`.

Tip: You can retrieve this array in your own code by calling the function `getMemberInfo()`, which returns this array.

Magic files in the hooks folder

You can create some files with specific names inside the hooks folder that your AppGini-generated application would use to perform a specific task. These files are optional, meaning that if they exist, your application will automatically use them to alter a default behavior. But if they don't exist, the default behavior will apply.

tablename-dv.js: modifying the behavior of the detail view through JavaScript

If you create a file in the hooks folder and name it `tablename-dv.js` (where *tablename* is the name of a table in your application), AppGini will automatically load that file and execute it as a JavaScript file in the browser whenever the detail view of the specified table is displayed. This is very useful to execute JavaScript code in the detail view.

For example, let's assume we have an *exams* table, and a score field in that table. We want to limit the contents of that field to a certain range of numbers, and warn the user if he enters a number outside that range. To do so, we could add some javascript code like the following in the magic `hooks/exams-dv.js` file.

```
$j(function() {
    $j('#score').on('change', function() {
        var score = parseInt($j('#score').val());
        if(isNaN(score) || score > 100 || score < 0){
            alert('Score must be between 0 and 100!');
            $j('#score').focus();
        }
    });
});
```

Line 1 in the code above makes sure this code won't be executed until the page content and jQuery are loaded to avoid triggering an error.

tablename-tv.js: modifying the behavior of a specific table through JavaScript

If you create a file in the hooks folder and name it `tablename-tv.js` (where *tablename* is the name of a table in your application), AppGini will automatically load that file and execute it as a JavaScript file in the browser whenever the specified table is displayed. This is very useful to modify the page content/layout or add custom behavior.

For an example of how this can be used to add new batch actions, please see the `batch_actions()` hook documentation.

Please note that despite the `-tv` suffix, this file is always loaded when the specific table is being viewed, whether the table view is being displayed or not. If you want to execute code *only if* the table view is visible, you can perform this check in the `tablename-tv.js` file:

```

$j() => {
  if(!$j('.table_view').length) return;

  // any code added below will be executed only in table view
});

```

tablename.fieldname.csv: changing the contents of options lists

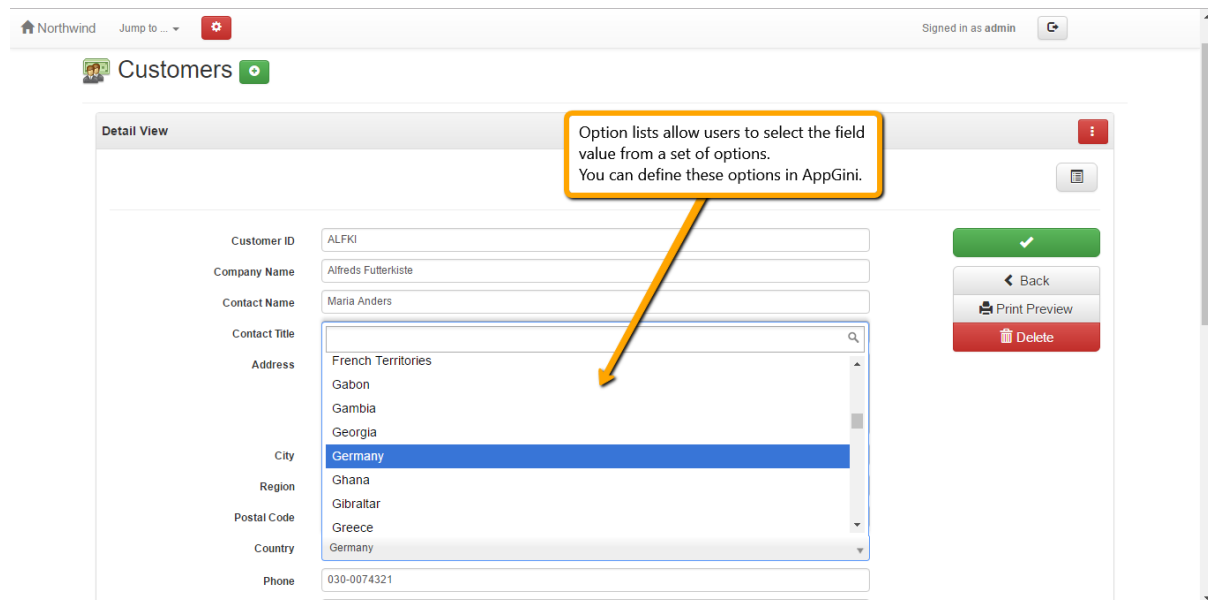


Figure 78: Option list example in an AppGini-generated application

In AppGini, you can define a field as an options list so that your application users can select the value of the field from a set of options. For example, the Country field in the screenshot to the left is an options list.

Now, what happens if you want to modify the contents of that options list, for example to limit the list to some countries and remove the others? Normally, you would have to open your project in AppGini, go to the Country field, modify the list contents, regenerate your application, and upload it. That's a long way to go.

So, we provide an easier method that doesn't involve regenerating the application. Simply, create a text file in the generated hooks folder and name it like this pattern: `tablename.fieldname.csv` .. For example, for the Countries list in the screenshot, the file should be named `customers.Country.csv`. Next, fill this file with all the options you want the user to be able to choose from, separated by double semi-colons. Here are the file contents for a choice of just 3 countries as an example:

```
United States;;United Kingdom;;France
```

It's now very easy to edit this file using any text editor to add/remove/modify options, without having to regenerate your application. However, please beware that this file takes precedence over the options provided in your AppGini project. So, if you decide later to modify the options in your project file and regenerate your application, you should either delete the `tablename.fieldname.csv` hook file or update it with the new options.

WindowMessages class

A class for displaying messages to the user on the next page load.

To add a message, use the `WindowMessages::add()` method:

```
WindowMessages::add('Hello world!', 'alert alert-success');
```

The first parameter is the message to display, and the second parameter is the CSS classes to apply to the message `<div>` container. The second parameter is optional, and defaults to `alert alert-info` if not specified.

The message(s) would be displayed on the next page load, and are specific to the current browser window. That is, if you have multiple browser windows open, each window will have its own set of messages.

To add a dismissable message, use the `WindowMessages::addDismissable()` method:

```
WindowMessages::addDismissable('Hello world!', 'alert alert-success');
```

Parameters are the same as the `WindowMessages::add()` method.

How does it work?

The `WindowMessages` class works by assigning a unique ID to each browser window. The ID is submitted with each request, and the messages are stored in the session under that ID. When the page is loaded, the messages are retrieved from the session and displayed.

The most significant part of this is that the messages are assigned to the current browser window, and not the current session. This means that if you have multiple browser windows open, each window will have its own set of messages.

This is particularly useful after inserting a new record, where the user is redirected to another page. In that case, if you'd like to display some message to the user, you can use the `WindowMessages` class to do so, and the message will be displayed on the redirected page. See the example in `tablename_after_insert()` hook documentation .

This is also useful for use in `tablename_after_update()` and `tablename_after_delete()` hooks, where you can display a message to the user after updating or deleting a record.

Including the window ID when redirecting to another page

Your AppGini application already includes the window ID when redirecting to another page, so you don't need to do anything special. But in case you're using your own code to redirect to another page, you need to include the window ID in the URL. You can do so by using the `WindowMessages::windowIdQuery()` method, like this:

```
$url = 'http://example.com/another-page.php?' . WindowMessages::windowIdQuery();  
redirect($url);
```

Including the window ID in a form in custom pages

Built-in AppGini pages already include the window ID in forms, so you don't need to do anything special. But in case you're using your own code to create a form in a custom page, you need to include the window

ID as a hidden field in the form. You can do so using this code where you want the hidden field in the form:

```
echo WindowMessages::includeWindowId();
```

How to display the messages?

Your AppGini application already includes the code to display the messages, so you don't need to do anything special. In case of custom pages however, you need to include the following code in your page where you want the messages to be displayed:

```
WindowMessages::getHtml();
```

Table and detail view classes

As of AppGini 5.60, a new advanced option was added to allow you to specify one or more space-separated CSS class names to apply to the table view and the detail view of a table. This option can be found by selecting a table in your project, then clicking the "Template" button. This would open a dialog as shown below.

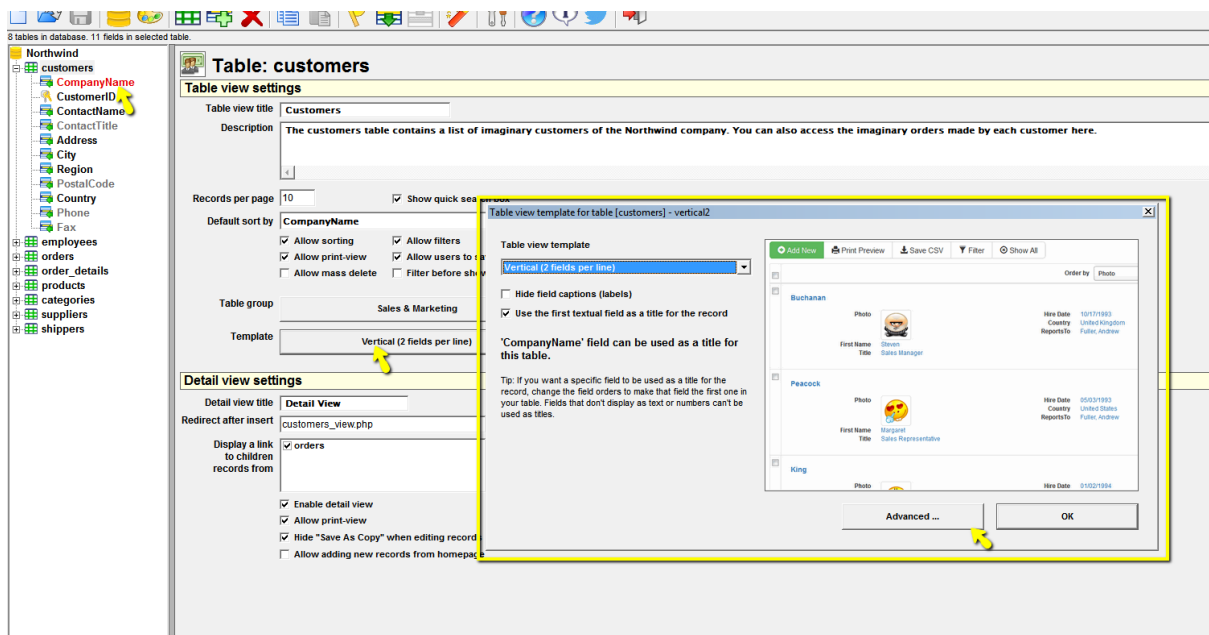


Figure 79: How to open the template window in AppGini 5.60 and higher.

Clicking the "Advanced" button will reveal even more options.

Adding custom limited-access pages and reports

In most applications, you might need to create additional customized pages besides the ones generated by AppGini. For example, you might want to add some reports, charts, switch boards, special forms, .. etc. In this article, we'll explain how you can create an additional page and limit access to it to authenticated users. We'll also explain how to integrate it as part of your AppGini application.

You probably want to achieve 3 goals while integrating new custom pages into your AppGini application:

1. **Control access to the page.** You want only authenticated users (or maybe only *some* authenticated users) to be able to access the page, while others are redirected to the homepage or the login form.
2. **Integrate the page appearance into your application.** That is, you want that custom page to display the same top navigation menu shown in the other pages of your application, and to have the same theme.
3. **Link to the page from other pages** so that your application users can easily find it. You might want to link to it from the homepage and/or from the “Jump to” drop-down menu in the top navigation bar.

We'll cover all the above points in this article.

Control access to your custom page

AppGini supports a membership system that is based on user groups.

1. You can grant some permissions to a group (or some groups), and all users under that group would automatically be granted those permissions.
2. Alternatively, you can grant some permissions only to a specific user rather than an entire group.
3. Another approach is to grant some permissions to any authenticated user regardless of which group they belong to.

Let's see how to apply any of these approaches to your custom page.

First of all, let's create a new file in the 'hooks' folder inside your AppGini-generated application folder. Let's call it “example.php”. Now, open that file in your text editor and paste the code below then save it.

```
<?php
define('PREPEND_PATH', '../');
$hooks_dir = __DIR__;
include("$hooks_dir/./lib.php");
```

The above code allows you to use the functions provided by AppGini in your custom page, including the function `getMemberInfo()` which you can use for checking permissions. Let's see how to implement each of the above access methods.

Grant access to one or more groups

In case you want all the users that belong to the “Admins” and “Data entry” groups (for example) to be able to access your custom page, let's edit the code to read like this

```

<?php
define('PREPEND_PATH', '../');
$hooks_dir = __DIR__;
include("$hooks_dir/./lib.php");

/* grant access to the groups 'Admins' and 'Data entry' */
$mi = getMemberInfo();
if(!in_array($mi['group'], ['Admins', 'Data entry'])) {
    echo "Access denied";
    exit;
}

echo "You can access this page!";

```

If you try accessing the above page from your browser while logged in as any user under the ‘Admins’ or ‘Data entry’ groups, you should see the message *You can access this page!* ... Otherwise, you should see the error *Access denied*.

Grant access to one or more users

Another case is when you want one or more specific users, rather than a whole group, to access the page. We’ll still use the `getMemberInfo()` function but the check will be slightly different:

```

<?php
define('PREPEND_PATH', '../');
$hooks_dir = __DIR__;
include("$hooks_dir/./lib.php");

/* grant access to the groups 'Admins' and 'Data entry' */
$mi = getMemberInfo();
if(!in_array($mi['username'], ['john.doe', 'jane.doe'])) {
    echo "Access denied";
    exit;
}

echo "You can access this page!";

```

If you try accessing the above page from your browser while logged in as the user ‘john.doe’ or ‘jane.doe’, you should see the message *You can access this page!* ... Otherwise, you should see the error *Access denied*.

Grant access to any logged user

Another case is to grant access to your page to all logged users. Here is the code for this scenario.

```

<?php
define('PREPEND_PATH', '../');
$hooks_dir = __DIR__;
include("$hooks_dir/./lib.php");

/* grant access to all logged users */
$mi = getMemberInfo();
if(!$mi['username'] || $mi['username'] == 'guest') {
    echo "Access denied";
    exit;
}

echo "You can access this page!";

```

The above will deny access to anonymous users and allow access to any logged user. If you’ve changed the default anonymous username of ‘guest’ in the admin area, you should update it in line 9 above.

Integrate the page appearance into your AppGini application

After controlling access to your custom page, the next step is to customize its appearance so that it matches the rest of the application pages. This can be very easily achieved by including the header and footer files as follows.

```
<?php
define('PREPEND_PATH', '../');
$hooks_dir = __DIR__;
include("$hooks_dir/./lib.php");

include_once("$hooks_dir/./header.php");

/* grant access to all logged users */
$mi = getMemberInfo();
if(!$mi['username'] || $mi['username'] == 'guest') {
    echo "Access denied";
    exit;
}

echo "You can access this page!";

include_once("$hooks_dir/./footer.php");
```

Link to the page from other pages

Finally, you want users to be able to easily reach your page. AppGini makes it easy to add links to the homepage and/or to the navigation menu. To do so, all you need to do is add a few lines to the “hooks/links-home.php” and/or “hooks/links-navmenu.php” files.

Tip! If you plan to add many custom pages to your application, it might not be very practical to place links to all of them directly into the navigation menu or the homepage. A more organized approach in this case is to create a page listing the custom links and add a link to that page rather than to each custom page.

If you are using AppGini versions before 5.90

If you’re using AppGini versions earlier than 5.90, you need to `include` language files when creating a custom page. In all of the above code snippets, change this part of the code:

```
include("$hooks_dir/./lib.php");
```

to:

```
include("$hooks_dir/./defaultLang.php");
include("$hooks_dir/./language.php");
include("$hooks_dir/./lib.php");
```

Third party resources used by AppGini applications

This is a list of third-party open source libraries and resources used by AppGini applications.

- **jQuery** Website, Documentation
- **Prototype** Website, Documentation (*will be removed in future releases*)
- **Scriptaculous** Website, Documentation (*will be removed in future releases*)
- **Lightbox2** Website and documentation (might be replaced or upgraded in future releases)
- **Initializr** Website
- **Bootstrap** Website, Documentation
- **Select2** Website and documentation
- **Datepicker** Website, Documentation
- **Bootstrap Timepicker** Website and documentation

Command-line parameters

You can use command-line parameters to automate AppGini. For example, you can add a command to generate an AppGini application as part of your deployment workflow in a batch/script file. Command-line parameters work only with AppGini Pro. Currently, the following parameters are supported:

--generate "path\to\project\file"

(Added in AppGini 5.11). Generates application from provided AXP project file, using the output path and file overwriting settings stored in the project file, then quits. If the path to the project file contains spaces, use double-quotes around the project path.

--output "path\to\output\folder"

(Added in AppGini 5.73). Specifies output path to use when generating an application, overriding the path stored in the project file. A project file must be specified using **--generate**. If the output path contains spaces, use double-quotes around the output path.

--log "path\to\log\file"

(Added in AppGini 5.81). Specifies a path to a log file to save the output log to when generating an app. If the log file path contains spaces, use double-quotes around it.

Examples

```
"C:\Program Files\AppGini\AppGini" --generate C:\projects\myapp.axp
```

This would launch AppGini and generate an application from myapp.axp to the folder stored in the project file.

```
"C:\Program Files\AppGini\AppGini" --generate C:\projects\myapp.axp --output C:\xampp\htdocs\myapp2
```

This would launch AppGini and generate an application from myapp.axp to the folder C:\xampp\htdocs\myapp2 overriding the folder stored in the project file if different. If the specified folder doesn't exist, AppGini will attempt to create it.

```
"C:\Program Files\AppGini\AppGini" --generate C:\projects\myapp.axp --log C:\logs\myapp.log
```

This would launch AppGini and generate an application from myapp.axp, and save output log to C:\logs\myapp.log.

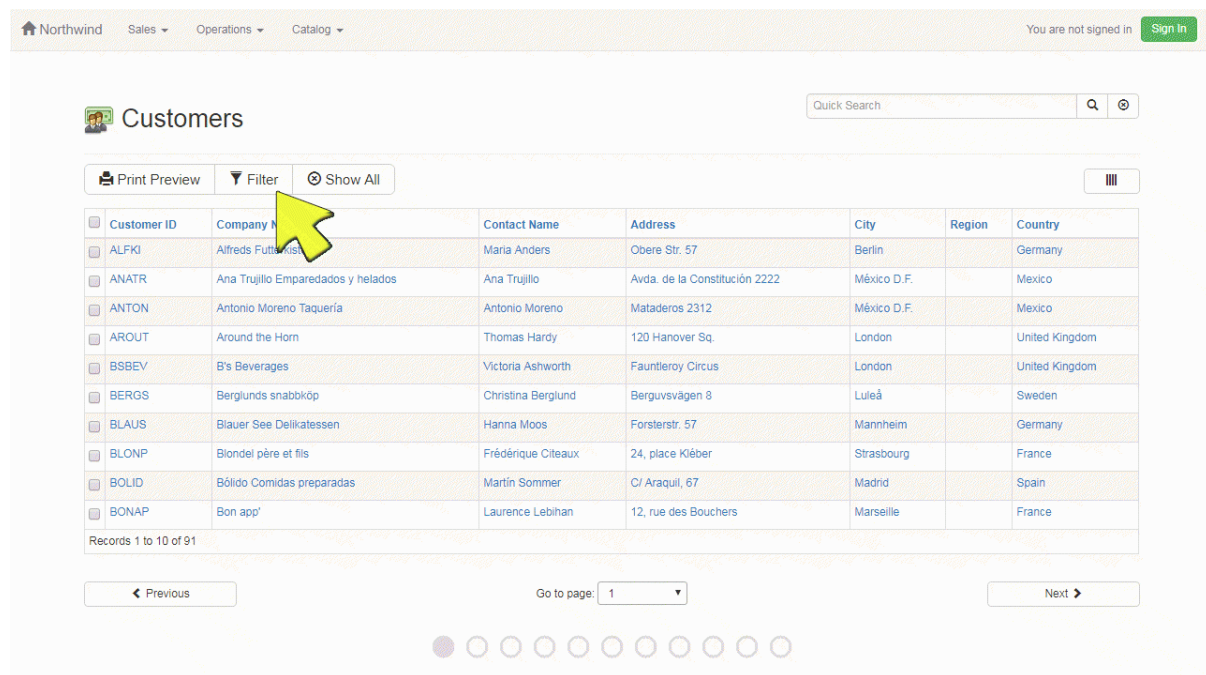
Special links and URL parameters

By default, the links to various tables in your AppGini-generated application pass no URL parameters, and thus display the default table view -- applying the default sorting and no filters. However, you can add special parameters to the URL that allow you to control how the table view data is displayed.

The default way to link to a table is to link to `tablename_view.php` (where `tablename` is the name of the concerned table). For example, if you have a table named `customers`, the default table view link would be `customers_view.php`.

There are some special URL parameters that you can use as part of the link to change what you are linking to. For example, if you want to link to the customers table, sorted by country, rather than not sorted, the link would be `customers_view.php?SortField=4` (assuming country is the fourth field in the table as it appears in your AppGini project). You can add other URL parameters to the link, separating them by `&` character (note that only the first parameter you add to the URL starts with a `?` character). You can add URL parameters in any order.

Here is a very easy way to obtain the table view link if you want to apply specific filters and sorting: select the concerned table in AppGini, then check the option **Allow users to save filters**. This would add a button down the filters page labeled **Save and apply filters**. You can then go to the filters page of your table, define the filters and sorting you wish to link to, and then click that button. This will display a *permalink* to the table view that you can copy and use to link to the filtered and sorted table. See the illustration below.



The screenshot shows the AppGini interface for the 'Customers' table. At the top, there's a navigation bar with 'Northwind', 'Sales', 'Operations', and 'Catalog'. A 'Sign In' button is on the right. Below the navigation bar, the 'Customers' table view is displayed. A 'Quick Search' bar is at the top right. Below the search bar, there are buttons for 'Print Preview', 'Filter', and 'Show All'. A yellow arrow points to the 'Filter' button. Below these buttons is a table with columns: Customer ID, Company Name, Contact Name, Address, City, Region, and Country. The table contains 10 rows of customer data. At the bottom, there's a pagination bar with 'Records 1 to 10 of 91', a 'Go to page:' dropdown set to '1', and 'Previous' and 'Next' buttons. Below the pagination bar is a row of 10 small circles, with the first one highlighted.

Customer ID	Company Name	Contact Name	Address	City	Region	Country
ALFKI	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin		Germany
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.		Mexico
ANTON	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.		Mexico
AROUT	Around the Horn	Thomas Hardy	120 Hanover Sq.	London		United Kingdom
BSBEV	B's Beverages	Victoria Ashworth	Fauntleroy Circus	London		United Kingdom
BERGS	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå		Sweden
BLAUS	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim		Germany
BLONP	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg		France
BOLID	Bólido Comidas preparadas	Martin Sommer	C/ Araquil, 67	Madrid		Spain
BONAP	Bon app'	Laurence Leblan	12, rue des Bouchers	Marseille		France

Figure 80: Obtain URL parameters from saved filters

You can then add this link to the homepage and/or the navigation menu for other users to access it without having to redefine the filters and sorting. To do so, just edit the generated `hooks/links-home.php` (for

adding the link to the homepage), and `hooks/links-navmenu.php` (for adding the link to the navigation menu).

Here is a table listing common URL parameters and their usage

SortField

- Default value: None
- Details: Specify the index of the field(s) to sort the table by. Examples:
 - `customers_view.php?SortField=4` sorts the table by the fourth field.
 - `customers_view.php?SortField=4,7` sorts the table by the fourth then the seventh field.

SortDirection

- Default value: `asc`
- Details: If a `SortField` is provided, `SortDirection` determines the direction of sorting. Possible values: `asc`, `desc`. Example:
 - `customers_view.php?SortField=4&SortDirection=desc` sorts the table by the fourth field in descending order.

FilterAnd[x]

- Default value: None
- Details: Please refer to working with filters programmatically

FilterField[x]

- Default value: None
- Details: Please refer to working with filters programmatically

FilterOperator[x]

- Default value: None
- Details: Please refer to working with filters programmatically

FilterValue[x]

- Default value: None
- Details: Please refer to working with filters programmatically

SearchString

- Default value: None
- Details: Displays records matching the provided search string. This is equivalent to typing a search term in the quick search box above the table view. Example:
 - `customers_view.php?SearchString=germany`

FirstRecord

- Default value: 1
- Details: This is equivalent to navigating the table view using the next/previous buttons or the *Go to page* drop-down. Thus, you could use it to jump to a specific page instead of the default first page of the table. Example:
 - `customers_view.php?FirstRecord=21` jumps to page 3 (assuming 10 records per page).

Print_x

- Default value: None
- Details: Used for displaying the print preview page. Example:

- `customers_view.php?Print_x=1` displays the print preview of the table view.

addNew_x

- Default value: None
- Details: If set to any non-zero value, displays the detail view for entering a new record if the user has insert permission to the table.

filterer_{fieldname}

- Default value: None
- Details: `{fieldname}` is the name of a non-autofill lookup field (that is, a lookup field displaying a drop-down rather than being automatically filled). You should use this with `addNew_x` to set a default value for the specified lookup field. Set the value of this parameter to the value of the primary key of the parent record. For example, to start a new order, setting the customer to the one whose ID is 203:
 - `orders_view.php?addNew_x=1&filterer_customer_id=203`

SelectedID

- Default value: None
- Details: If provided, displays the detail view for editing the record identified by the given primary key value. If the user doesn't have edit permission for the given record, a read-only detail view is displayed. Example:
 - `customers_view.php?SelectedID=203` displays the detail view for editing the customer whose primary key is 203.

dvprint_x

- Default value: None
- Details: If set to any non-zero value, and a `SelectedID` value is also provided, displays the print preview of the detail view of the record specified. Example:
 - `customers_view.php?SelectedID=203&dvprint_x=1`

delete_x

- Default value: None
- Details: If set to any non-zero value, and a `SelectedID` value is also provided, the specified record is *deleted* if the user has permission. > **Use this with extreme caution!**

noQuickSearchFocus

- Default value: None
- Details: AppGini 5.90 and above
 - Passing `noQuickSearchFocus=1` when linking to a table view prevents auto-focusing of quick search button. This is useful for example when you embed a table view in another page and don't want the browser to "jump" down to the embedded page.

From Data to Dashboards, A Guide to Redash Integration with AppGini

Table of Contents

- What is Redash?
- Installing Redash on your server
- Connecting Redash to your AppGini application
- Creating a query in Redash
- Creating a dashboard in Redash
- Creating alerts in Redash
- Dive deeper into Redash
- Performance considerations when using Redash
- Conclusion

What is Redash?

Redash is an open-source data visualization and dashboarding tool that allows users to connect to various data sources, create interactive visualizations, and build dynamic dashboards. It provides a user-friendly interface for querying and exploring data, making it easier for non-technical users to access and analyze data.

Redash supports a wide range of data sources, including relational databases (e.g. MySQL databases used in AppGini applications), NoSQL databases, cloud storage services, and APIs. It allows users to write queries using SQL or other query languages specific to the data source and visualize the results in different chart types such as bar charts, line charts, pie charts, and more.

Installing Redash on your server

Redash is a stack of several components, including a Python web application, a PostgreSQL database, a Redis server, and several other components. It can be installed on a Linux server using Docker, or on a Windows server using a virtual machine. The installation process is documented in detail in the Redash documentation.

Setting up Redash can be a bit challenging, especially if you're not familiar with Docker. If you're not comfortable with the installation process, we can help you set up Redash for a small fee. Please contact us for details.

Connecting Redash to your AppGini application

Once you have Redash installed and running, you can connect it to your AppGini application. To do this, you'll need to create a MySQL user *with read-only access* to your AppGini database. Why read-only access? Because Redash will be running read-only queries against your database, and it's a good idea to limit its access to avoid any accidental changes to your data.

To create a read-only user, you can use the following SQL query in phpMyAdmin or any other MySQL client:



Figure 81: War room!

```
CREATE USER 'redash'@'%' IDENTIFIED BY 'strong-password';
GRANT SELECT ON `your-database-name`.* TO 'redash'@'%';
```

Replace `strong-password` with a strong password of your choice, and `your-database-name` with the name of your AppGini database. If you're not sure about the name of your database, you can find it in the `config.php` file in your AppGini application's folder.

Once you've created the user, you can connect Redash to your AppGini database by following the steps below:

1. Log in to Redash as an admin user.
2. Click on the "Settings" link in the bottom-left corner of the page.
3. Click on "Data Sources" in the left sidebar, then click on the "New Data Source" button.
4. In the search box, type "MySQL", then click on the "MySQL" data source.
5. You'll be asked to provide a name for the data source. You can use "AppGini" or any other name you like.
6. In the "Host" field, enter the IP address or hostname of your MySQL server. If you're running Redash on the same server as your AppGini application, you can usually use `localhost`.
7. Leave the "Port" field empty unless you're using a non-standard port for MySQL.
8. Provide the username and password you created earlier.
9. If your MySQL server is configured to use SSL, you can enable SSL by checking the "Use SSL" checkbox.
10. Click on the "Create" button to save the data source.
11. Next, click the "Test Connection" button to make sure Redash can connect to your AppGini database.
12. If the connection test is successful, click on the "Save" button to save the data source.

Here is a screencast showing the above steps

Your browser does not support the video tag.

Creating a query in Redash

Once you've connected Redash to your AppGini database, you can start creating queries. To create a new query, follow these steps:

1. Click on the "Queries" link in the left sidebar.
2. Click on the "New Query" button.
3. Select the "AppGini" data source you created earlier from the "Data Source" dropdown at the top left of the page.
4. In the "Query" field, enter your SQL query. You can use the "Schema" tree on the left to browse the tables and fields in your database and click the » button next to a table or field to insert it into your query.
5. Click on the "Execute" button to run the query and see the results.
6. If the query runs successfully, click on the "Save" button to save the query. You should give it a name that describes what the query does.
7. You can optionally click on the "Visualize" button to create a visualization of the query results. You can choose from a variety of chart types, including bar charts, line charts, pie charts, and more.
8. Once you're done, click on the "Save" button to save the visualization.
9. In order to be able to display the query results or visualization in a dashboard, you should click the "Publish" button. This will also make the query or visualization available to other users of Redash.

Here is a screencast showing the above steps

Your browser does not support the video tag.

HINT! You can use the DataTalk plugin for AppGini to create the queries you need in Redash.

The plugin allows you to create queries by just describing them in plain English. Here is a video showing how it works:

Your browser does not support the video tag.

DataTalk plugin saves you the hassle of writing complex SQL queries. And you don't need to remember the names of tables and fields in your database or even care about misspelling them. Just describe what you need in plain English and the plugin will take care of the rest!

Creating a dashboard in Redash

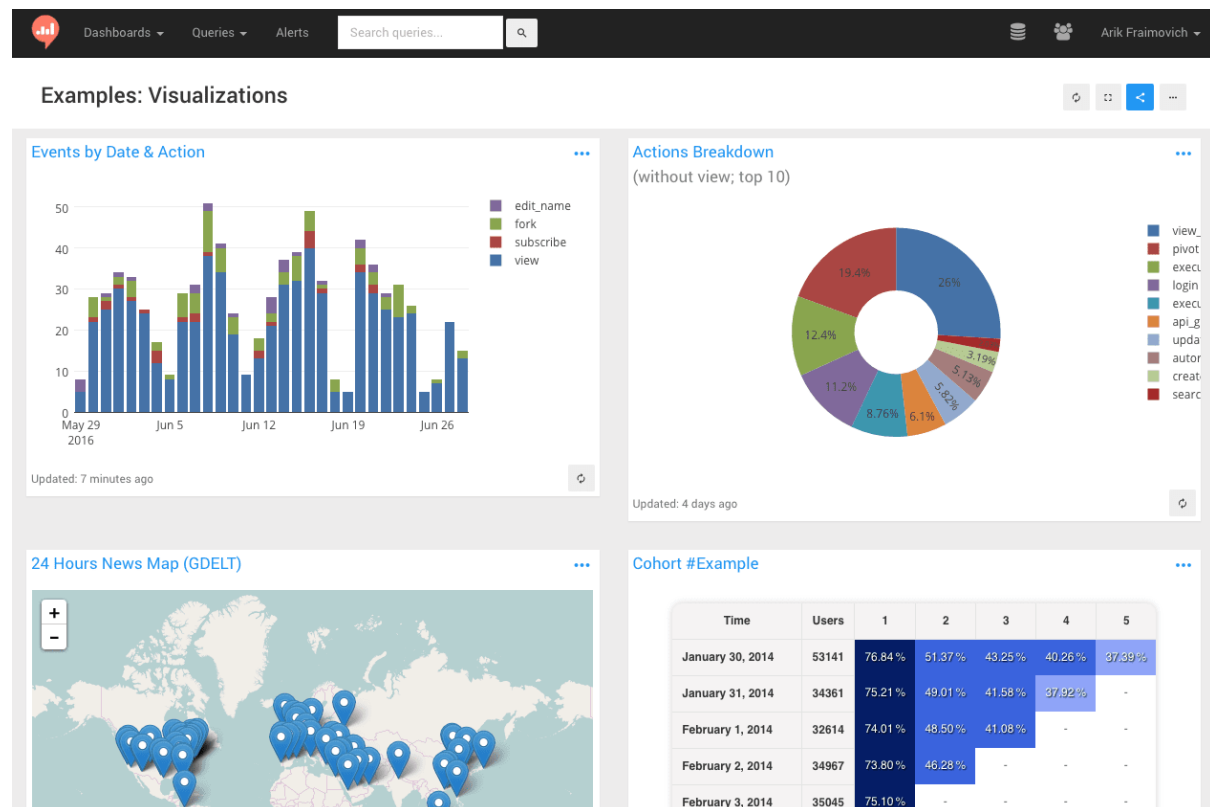


Figure 82: Redash dashboard example

Dashboards in Redash are made up of widgets. Each widget can display the results of a query or a visualization. To create a dashboard, follow these steps:

1. Click on the “Dashboards” link in the left sidebar.
2. Click on the “New Dashboard” button.
3. Give your dashboard a name and click on the “Save” button.
4. Click on the “Add Widget” button.
5. Select the query you want to display in the widget from the “Query” dropdown.
6. Select the visualization type you want to use from the “Visualization” dropdown.
7. Click on the “Add to Dashboard” button to add the widget to your dashboard.
8. You can move and resize the widget to control its position and size on the dashboard.
9. Repeat steps 4-8 to add more widgets to your dashboard.
10. Finally, click on the “Done Editing” button to save the dashboard.
11. If you want to share the dashboard with other Redash users, you can click on the “Publish” button.

Creating alerts in Redash

Redash allows you to create alerts that will be triggered when the results of a query meet certain conditions. For example, you can create an alert that will send an email to a user when the number of orders in the last 24 hours exceeds a certain threshold. To create an alert, you should configure the underlying query to automatically refresh periodically. You also need to configure alert destinations, which can be email addresses, Slack channels, webhooks, or several other destinations.

Here is an example of an alert that sends an email to the sales team when the number of orders in the last 7 days is zero:

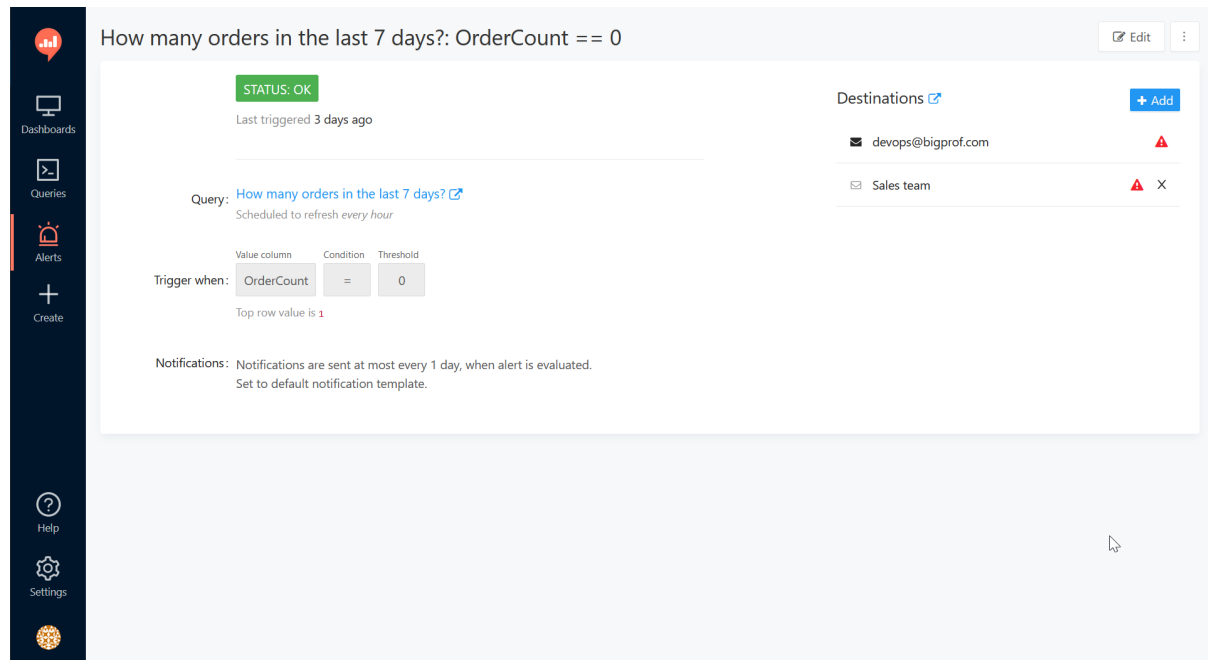


Figure 83: Redash alert example

Dive deeper into Redash

Redash has a very informative video playlist on YouTube that goes into more details about creating queries and dashboards. We've included the videos below for your convenience:

Performance considerations when using Redash

Redash is a great tool for visualizing data from your AppGini application. However, users can create complex queries that can put a heavy load on your database server. Moreover, Redash allows users to create queries and dashboards that auto refresh periodically – sometimes as frequently as every minute or so. This can put a heavy load on your database server, especially if you have a large number of users.

One way to avoid this, without limiting the functionality of Redash, is to set up a separate read replica of your MySQL database and connect Redash to it instead of the main database. This way, Redash users will be querying the read replica instead of the main database, and this will not affect the performance of your AppGini application.

Conclusion

Throughout this tutorial, we've covered the integration of Redash with AppGini to unlock advanced data visualization and dashboarding capabilities. You've learned how to set up Redash on your server, connect it to your AppGini database with enhanced security through read-only access, and utilize the DataTalk plugin to facilitate query creation without deep SQL knowledge. The step-by-step guidance provided should now empower you to create insightful visualizations, build interactive dashboards, and share your findings with ease.

As you apply these new skills, keep in mind the performance considerations vital for maintaining your application's responsiveness—especially the strategy of employing a read replica for your database to mitigate the load from complex and frequent queries. With Redash as your tool of choice, you are well-positioned to elevate the data experience for your team and stakeholders, ensuring your data is not just informative but also actionable.

I see an error or a blank page in my AppGini app - how to troubleshoot?

First of all: Have you made a customization to your AppGini app, either through hooks , or by directly editing some of the generated files outside hooks? If yes:

1. Open your AXP project file in AppGini.
2. Generate the app to a **new empty folder** . This would create a clean app, without any customizations.
3. Run your new app in your browser, trying to reproduce the error. If the error doesn't occur, then we know there is something wrong in the customization you've made.
4. You can either undo the customization, or if you want to debug it further, keep reading below for steps to discover the error and debug it.

The steps below are helpful to find the exact cause of the error, whether it's in your customized code, or in the generated code of your app.

- Please press F12 to open the browser's inspector, then click on the 'Console' tab. Is there any error reported there?

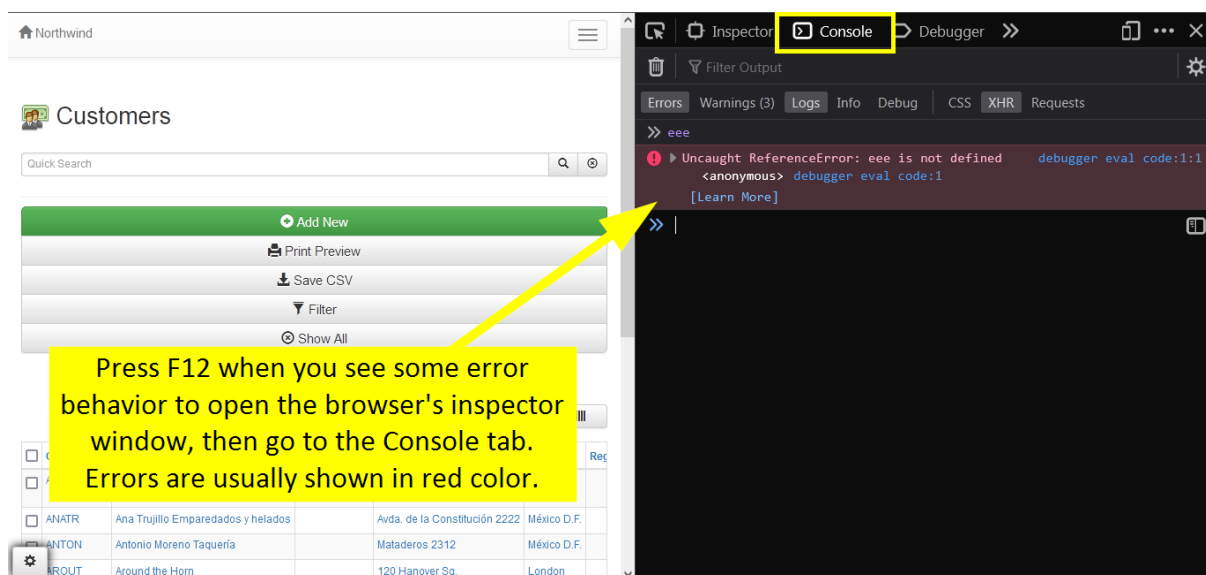


Figure 84: Inspector console tab showing an example error for an AppGini app.

- If not, and if you are able to access the admin area:
 - Go to the *Utilities* menu > *View/Rebuild fields* and perform the suggested fixes, if any.
 - If no issues reported in *View/Rebuild fields* page, go to *Utilities* menu again > *Query logs* . Are there any errors reported? If yes and any of the following applies:
 - * You've made a customization to the generated code, or
 - * you have calculated fields in your application, or

Admin Area

Groups

Members

Utilities

User's area

Sign Out

View/Rebuild fields

Show all fields

This page compares the tables and fields structure/schema as designed in AppGini to the actual database structure and allows you to fix any deviations.

Found 0 non-existing fields that need to be created.
Found 1 deviating fields that might need to be updated.
It's highly recommended that you create a database backup first before attempting to make any fixes here.

Field	AppGini definition	Current definition in the database	
Customers			
Employees			
Orders			
Order Items			
Products			
Product Categories			
✖ Picture	VARCHAR(40) NULL	VARCHAR(100)	⚙️ Fix it
Suppliers			
Shippers			

Found 0 non-existing fields that need to be created.
Found 1 deviating fields that might need to be updated.
It's highly recommended that you create a database backup first before attempting to make any fixes here.

Admin area > Utilities menu > View/Rebuild fields: if you see any errors reported, try fixing them. You can fix an individual error through the 'Fix it' button to the right of it, or 'Fix all fields' at once through the button on the top.

Figure 85: View/Rebuild fields page showing a fix button

* you have one or more custom SQL queries for lookup fields, if so, try revising your code to make sure the SQL queries are valid. You can use the ‘Interactive SQL queries tool’ under the *Utilities* menu to test your query. Or you can use 3rd party tools like phpMyAdmin.

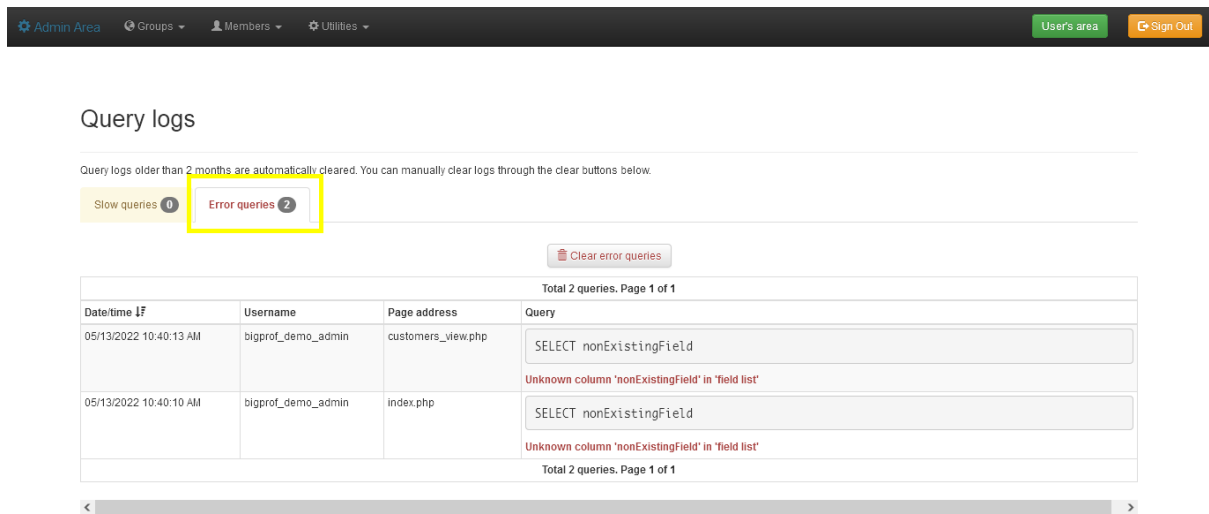


Figure 86: Query logs page showing example SQL errors

- If you found no SQL errors, open your server **error_log** file and check to see if any errors related to your AppGini app are reported in it.
 - The location of that file varies based on your web server software and your OS. You should consult your web server documentation to learn where to find the error log file. For example, for Apache on linux, that file is *typically* at `/var/log/apache2/error.log` (but that might vary based on your environment configuration).

If you find one or more errors reported in the error log file, and you’ve made a customization to the generated code, try to revise your code and fix any syntax errors.

- If none of the above applies to your AppGini application, please send us your AXP project file along with any error messages, screenshots and/or any other details about the error and how to reproduce it. You can send these via the contact form at <https://bigprof.com/appgini/support-request>

Useful links

Setting up a test environment for your web applications

You can test your AppGini-generated web applications on a local machine before deploying them online. To do so, you need to install a web server , MySQL , and PHP . Of course, installing and configuring all of these programs is a lot of headache. Fortunately, there is an easier way: download and install Xampp , a single download that takes care of all the necessary work in one shot.

Note: All of the above tools are free and open source software.

Uploading the generated code to your server

To upload the generated code to your server, you should use an FTP client. A very powerful program is FileZilla .

Our online course on customizing AppGini applications

We prepared a course for power users of AppGini, who'd like to add more functionality to their applications by writing custom code. Through 4.5 hours of video, you'll finish 30 practical examples, featuring more than 500 lines of code, that cover the most common hook questions we receive from users. Preview the course.

Hosting providers

Most hosting providers support hosting the applications generated by AppGini out-of-the-box, since AppGini generates PHP applications that connect to MySQL databases. PHP and MySQL are very widely available through almost any hosting provider.

If you're looking for a very easy to use and **highly reliable cloud hosting service** , we've been using Digital Ocean for many years and are quite content with their simplicity and reliability. This special link provides you with \$100 credit to try their service.

Digital Ocean however requires some experience managing your server updates and software installations (they have many detailed guides for that though). If you're looking for managed hosting (where the provider takes care of installing, securing and upgrading your server software, as well as resolving the majority of technical issues), Bluehost is a very good option.

A good source of information regarding hosting providers is the webhostingtalk forum .

Code management

If you plan to manually edit the generated code, you should be careful to avoid overwriting your modifications in case you regenerate the code later. CVS software helps greatly with organizing and versioning your code. It makes it very easy to undo (revert) harmful changes, and merge your modifications into newly generated code. One such great, easy and free program is TortoiseGit . We have a tutorial on how to use it with AppGini code .

Reference material

- [HTML reference](#)

- [CSS reference](#)
- [Bootstrap 3 reference](#)
- [jQuery API reference](#)
- [PHP reference](#)
- [MySQL SQL reference](#)

Contribute to AppGini documentation

The content of this documentation is hosted on GitHub and is open-source. This means you can contribute to it by submitting pull requests to the GitHub repository. This can be done very easily by clicking the “Edit this page on GitHub” link at the top of each page.

The following video shows how to contribute to the documentation:

By contributing to the documentation, you help other users and make AppGini better for everyone. Thank you for your help!