

Introduction

What is AppGini?

AppGini is a program that helps you develop user-friendly web database front-ends rapidly. You do not need to have any programming background to use it. Just define your database, set some options, click the Generate button, and ... that's it !

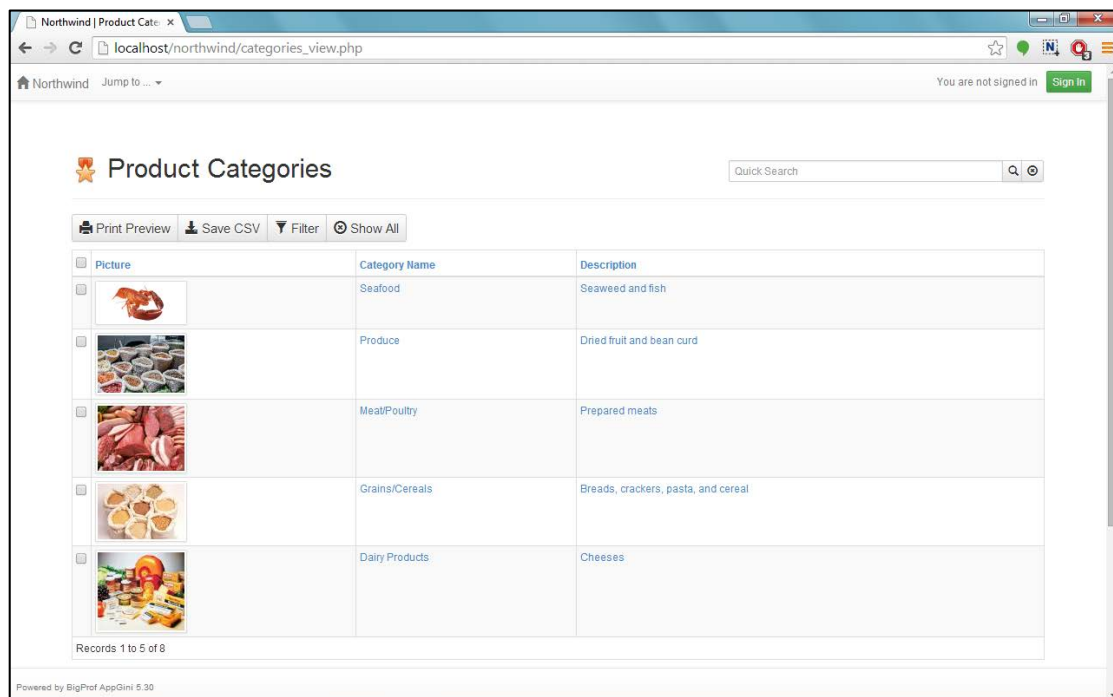
AppGini helps you create professional web database front-ends ready for uploading to your website. It has all the powerful features you would expect from a professional database application: data navigation, sorting, filtering, editing, inserting, deleting, importing, exporting, and users/groups management .

Moreover, there are many other features that help enhance your application. Features like foreign key functionality (lookup fields), full control over the application's appearance and behavior, support for image and file uploads, and many others ...

What languages and databases are supported by AppGini?

AppGini generates applications written in PHP language that connect to MySQL databases. Again, you never need to know anything about PHP or MySQL to use AppGini. PHP and MySQL are very widely supported by Internet developers and web hosting providers. So, you'll probably get the generated applications to work with your server without any pre-configuration at all .

As you read on, or, if you prefer, experiment on your own with AppGini, you'll discover all the great features built into it. So ... please follow me!



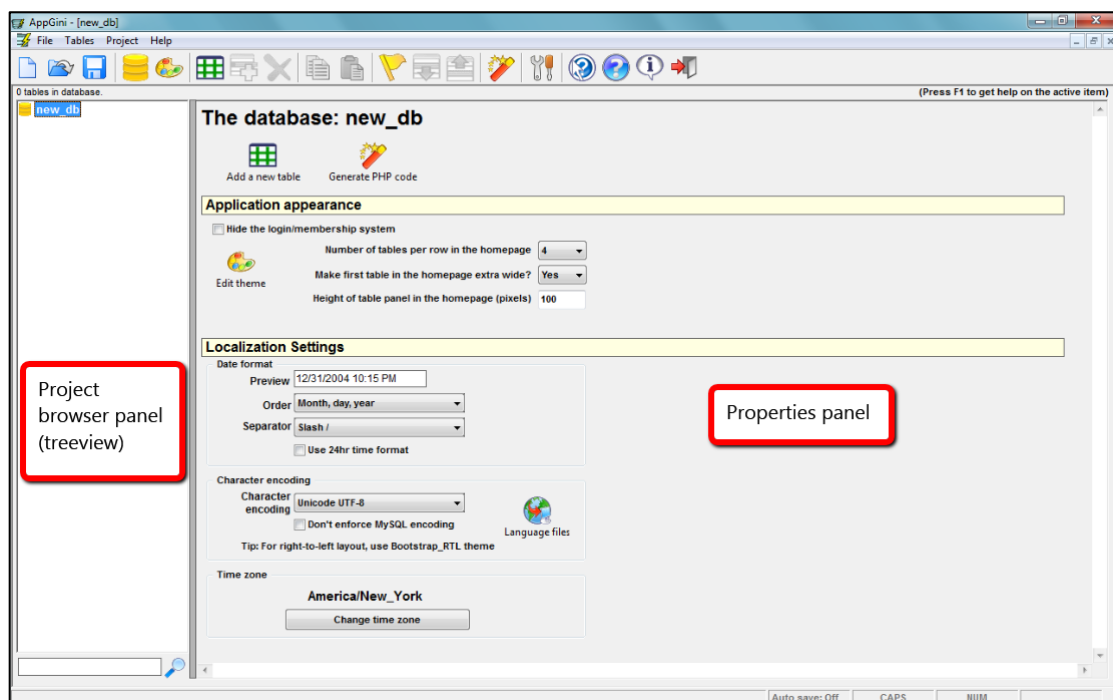
Working with projects

What is an AppGini project?

An AppGini project is the starting point for your work. The project is where you describe and design your database tables and fields, and define your application's appearance and behavior. After you finish working with your project, you just click the Generate button on the toolbar or press F5. The Generate button on the toolbar. AppGini then generates all the PHP code for your application and saves it to a folder of your choice. You can then upload the generated files to your web server using any FTP client.

How do I start a new project?

Start AppGini, then from the File menu, choose New. You can also click the 'New Project' icon on the tool bar. A new project window will appear, as shown below. The database has a default name of 'new_db'. To change it, click on the database name in the left panel, then press F2 and type the new name. This name is used for identification only and is not related to the actual database name on your server, which you can configure later during generated application setup.



The project window

This is your working area. This window has two panels: the Project Browser panel at the left lets you view your project contents (the database, tables and fields) in an easy to navigate hierarchical manner. The right panel is the properties area. When you click on any item in the Project Browser panel, its properties are displayed in the properties panel.

In the professional version of AppGini, projects can be saved as .AXP files and opened later. The freeware version can open project files but cannot save changes to them.

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.



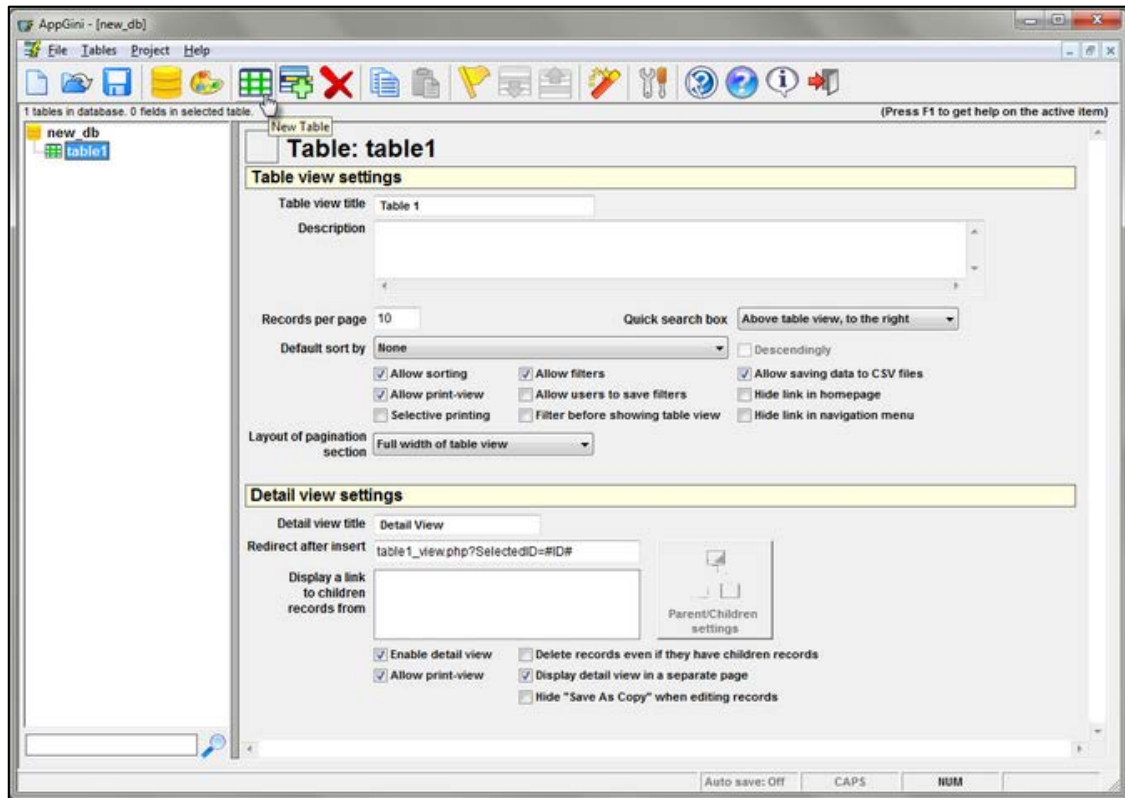
To obtain context help on any button or property, press F1 while it has the focus. You can use the Tab key to move the focus between all the elements of the project window .



To obtain context help on an icon on the tool bar, click on the help pointer icon and click with the mouse on the icon that you need help about. To stop displaying help on icons, click again on the help pointer icon.

Working with tables

How can I add a table to a project? Make sure you have an open project, and then click on the 'New Table' icon on the tool bar, or from the Tables menu, select 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

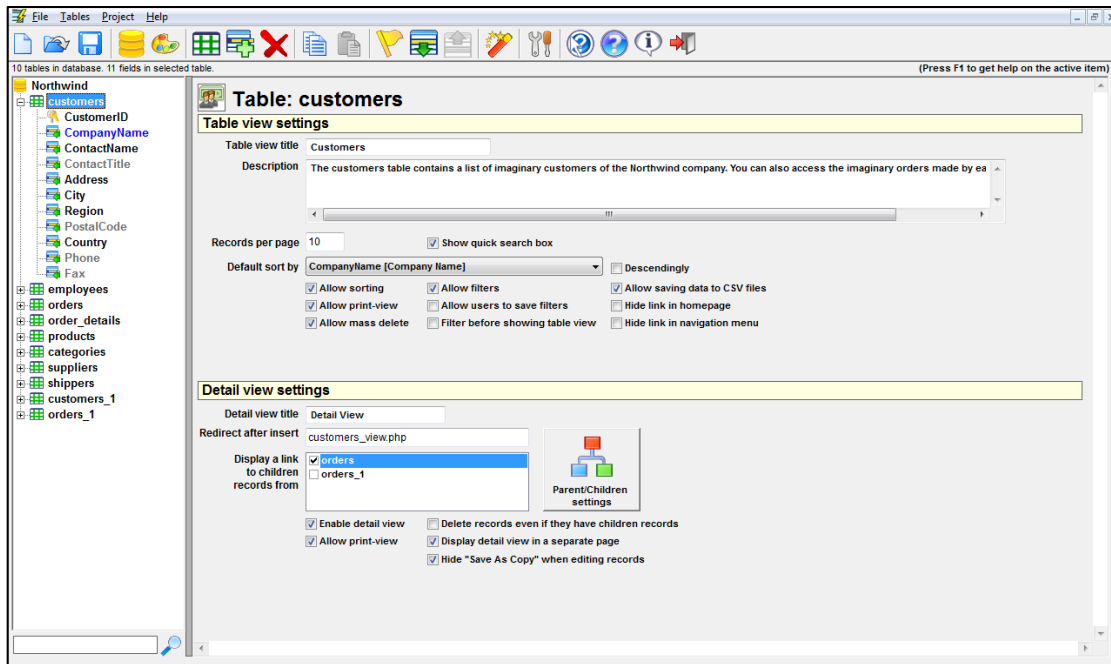
Can I use a table from another AppGini project?

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



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.

Tables menu

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

Project menu

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

Help menu

Includes various help resources

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.

Toolbar > Open Project

Click on this icon to open a previously saved project.

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.

Toolbar > Project Properties

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

Toolbar > Application Theme

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

Toolbar > New Table

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

Toolbar > New Field

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

Toolbar > Delete Selected Table/Field

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

Copy

Copies the selected table or field to the clipboard

Paste

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

Toggle Highlight

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

Toolbar > Move Table/Field up

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

Toolbar > Move Table/Field up

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

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.

Toolbar > Preferences

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

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.

Toolbar > Help

Click on this icon to view the online help.

Toolbar > About AppGini

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

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.
Table view and detail view

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.

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.

Table view and detail view

This is a screen shot that demonstrates an example of how the table view and the detail view would look like after generating your application.

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.

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.

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.

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.

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.

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.

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.

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).

Table properties > Allow print-view

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

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.

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.

Table properties > Allow mass delete

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

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.

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.

Table view and detail view

This is a screen shot that demonstrates an example of how the table view and the detail view would look like after generating your application.

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.

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

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.

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.

Table properties > Enable detail view

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

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. If you check this option, the above check will still be performed, but the user will be prompted to confirm deletion of the record. If the user confirms, the record will be deleted. In this case, children records will not be deleted, but will have no parent record.

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.

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.

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.

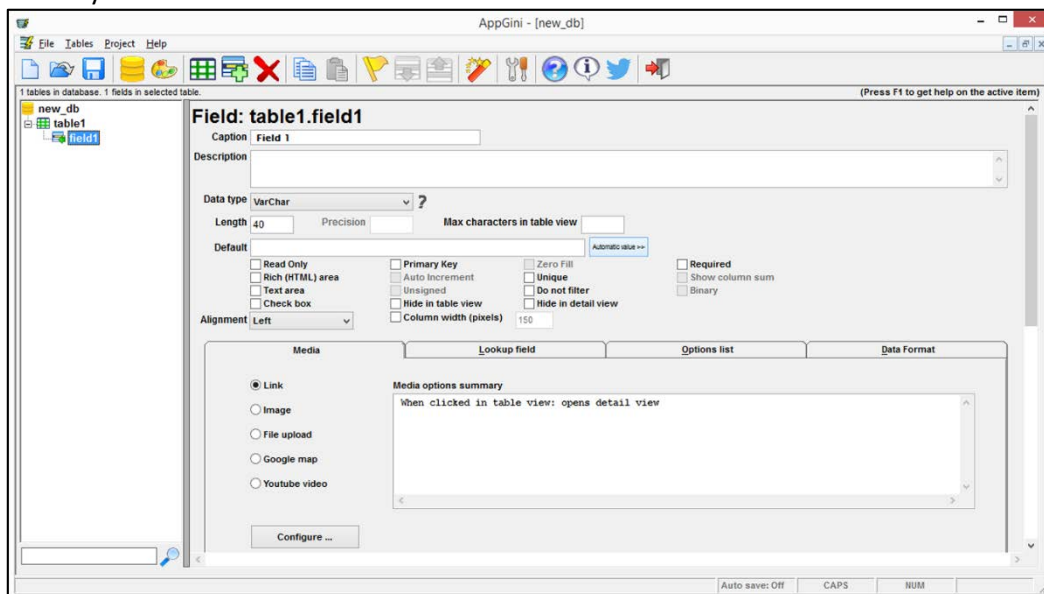
Search box

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.

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.



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.

Another way is to add a new field to one of your tables, and in the field properties panel, open the drop down labeled "Copy properties from" and select the field that you want to copy. The properties of the new field will then be set to the same values as those of the field you selected from the drop down .

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?

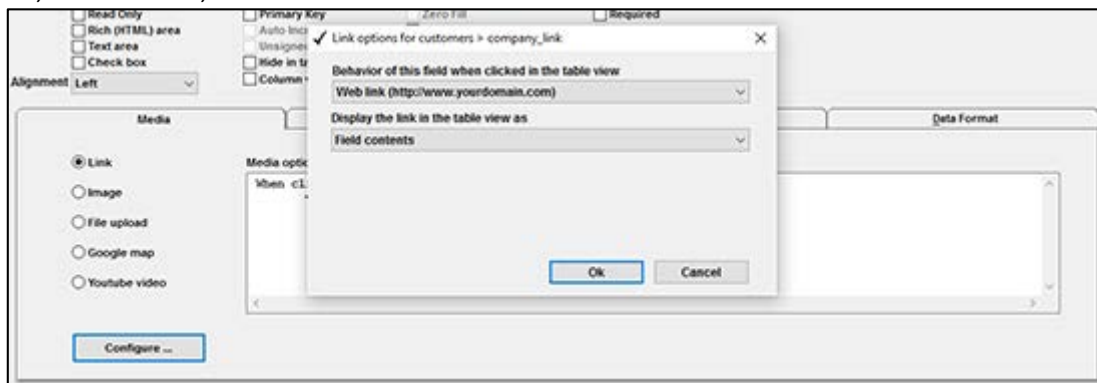
In AppGini, click with the mouse on any property and press F1 to obtain help about its function.

The Media Tab

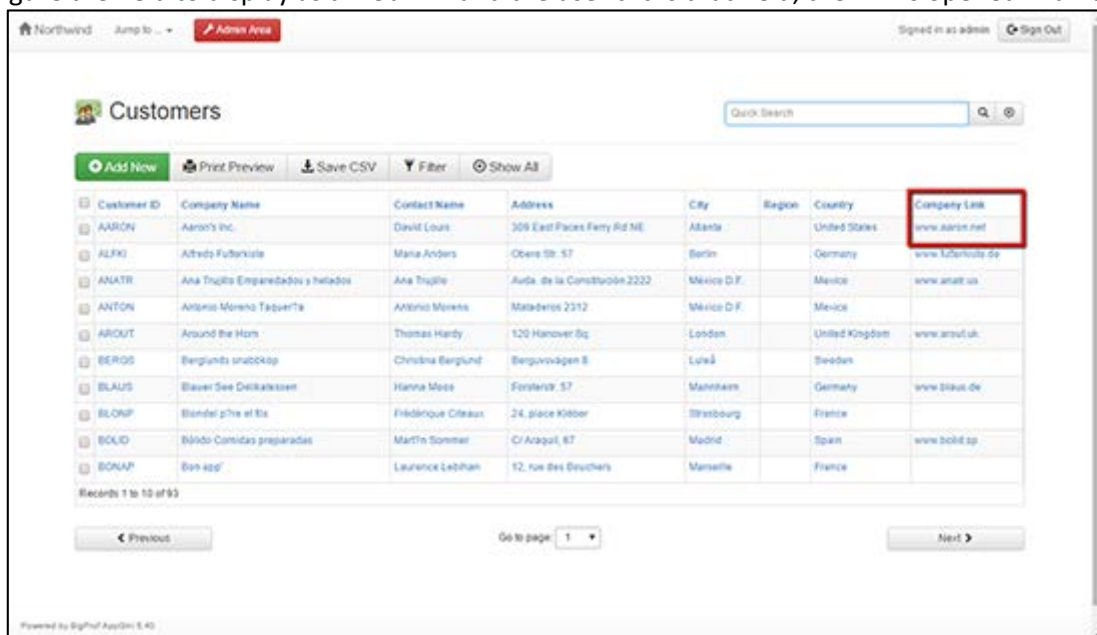
This tab allows you to configure your field to be displayed as a web-link, an image, a file, a google map or even a YouTube video.

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.

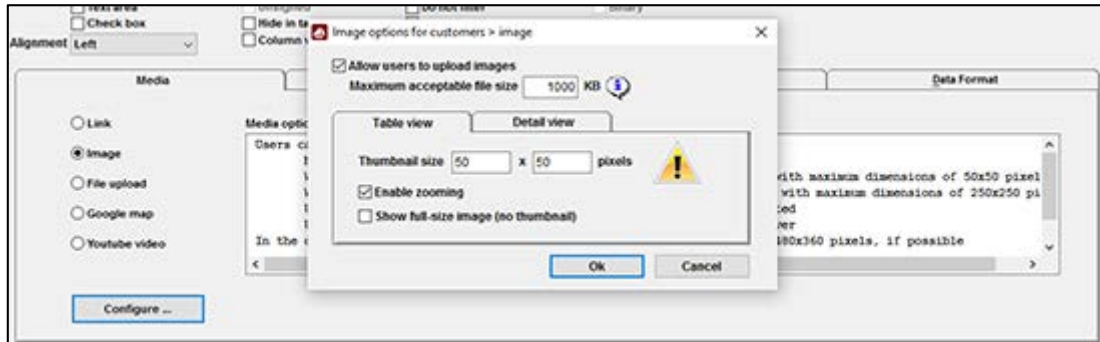


If you configure the field to display as a web-link and the user clicks that field, the link is opened in a new window.

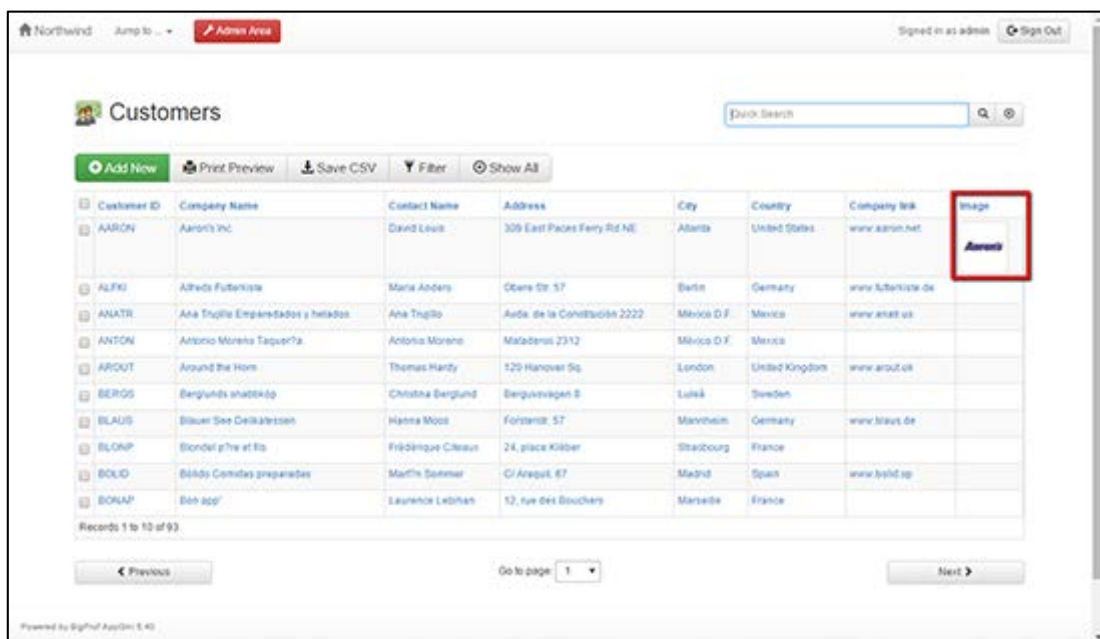


The Image option

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.

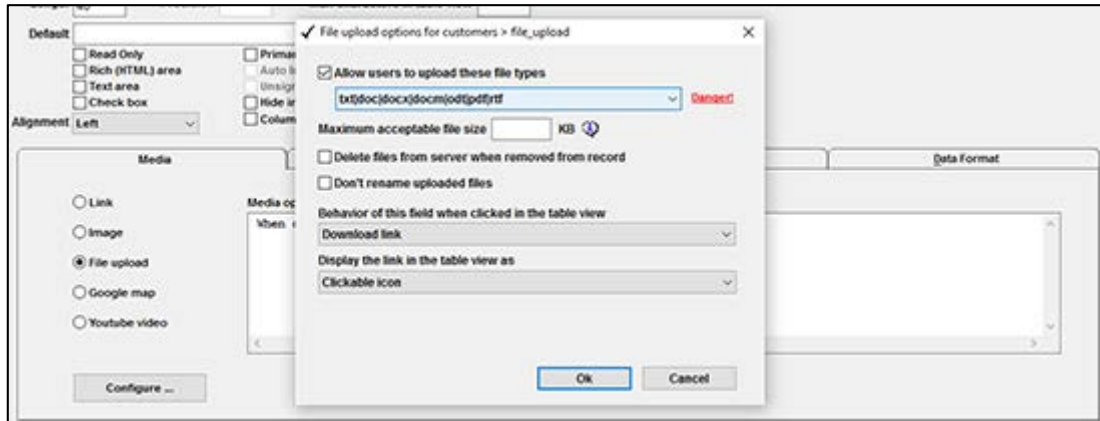


You can choose how to display the image. It can be displayed as a zoomable thumbnail image in the table and detail view. You can also configure the thumbnail size.

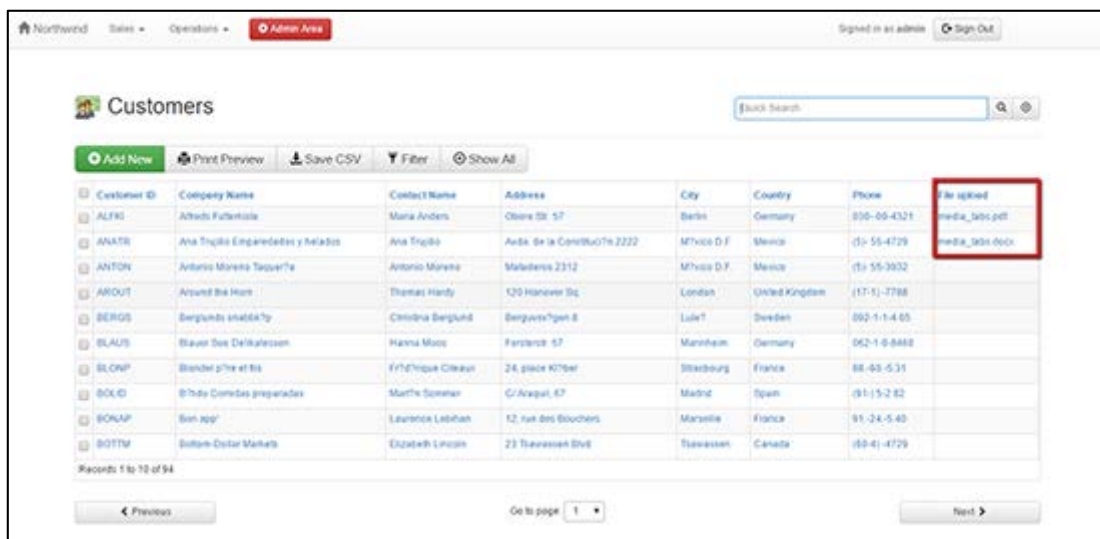


The File upload option

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.

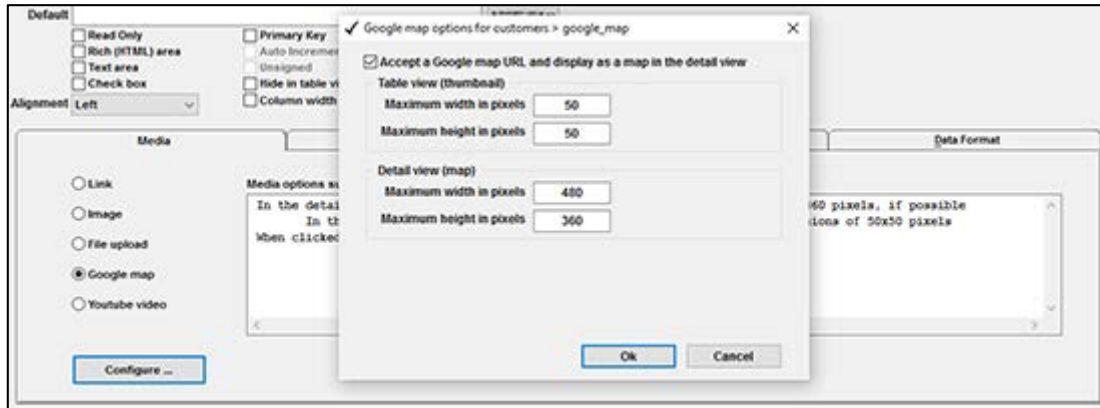


You can configure the field to be displayed as the field content, clickable icon or contents of another page.

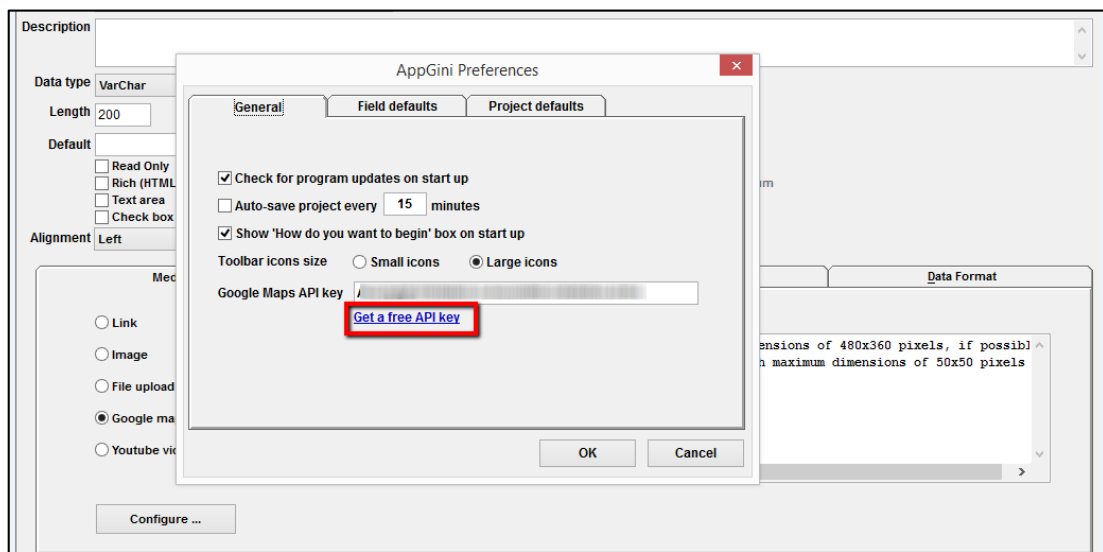


Google Maps

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.

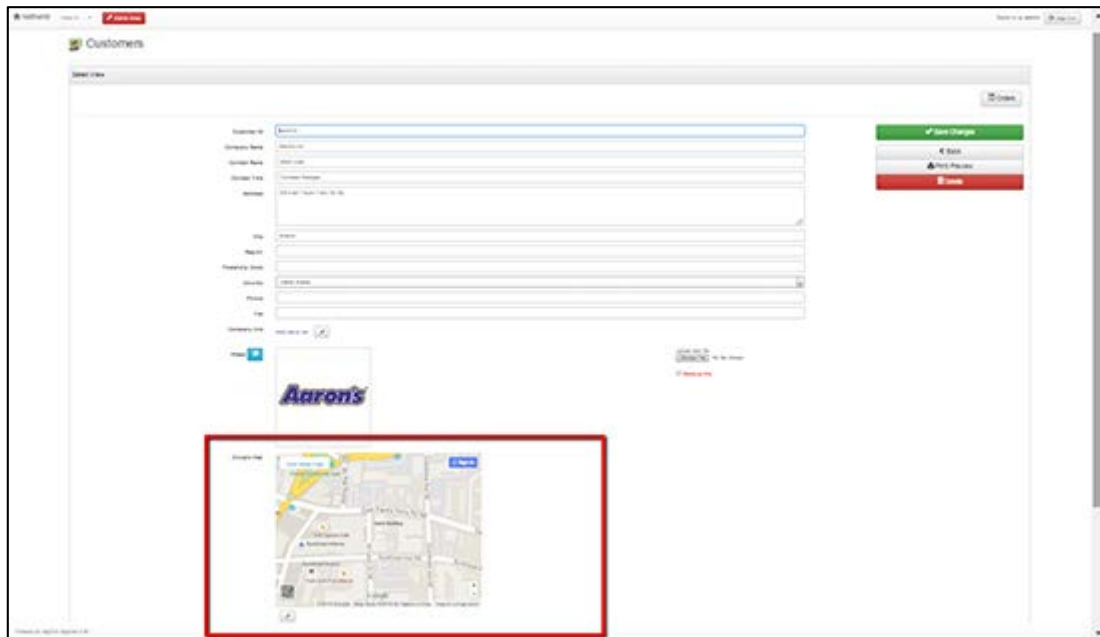


Google Maps require a Google API key to work correctly. You can add one by simply clicking the settings dialog to get a key. This allows you to insert interactive maps into your application.

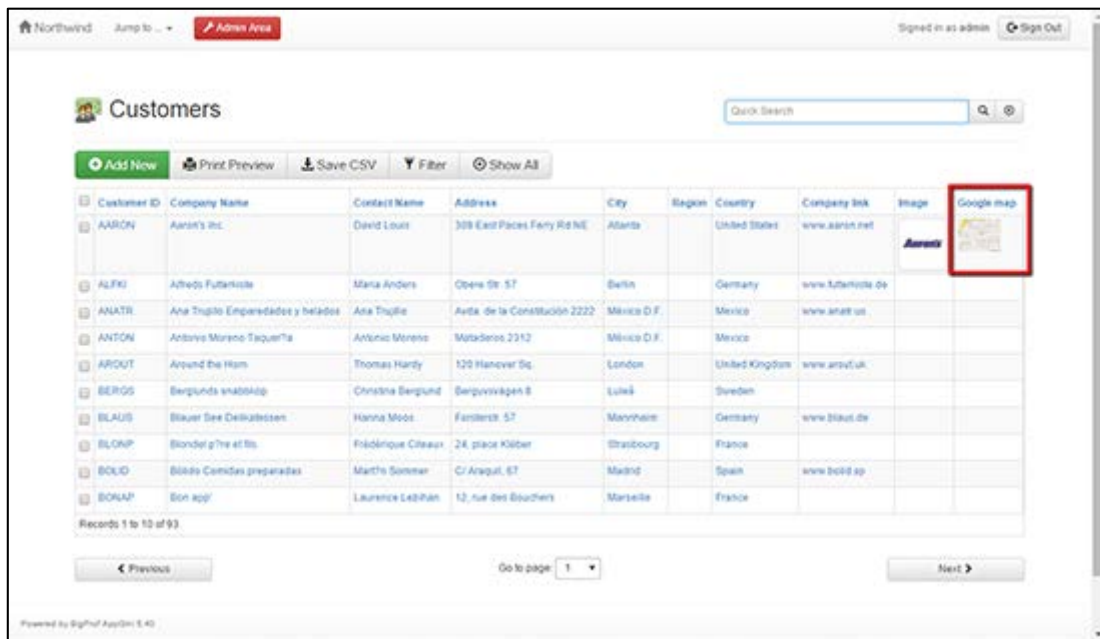


APPGINI DOCUMENTATION

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.

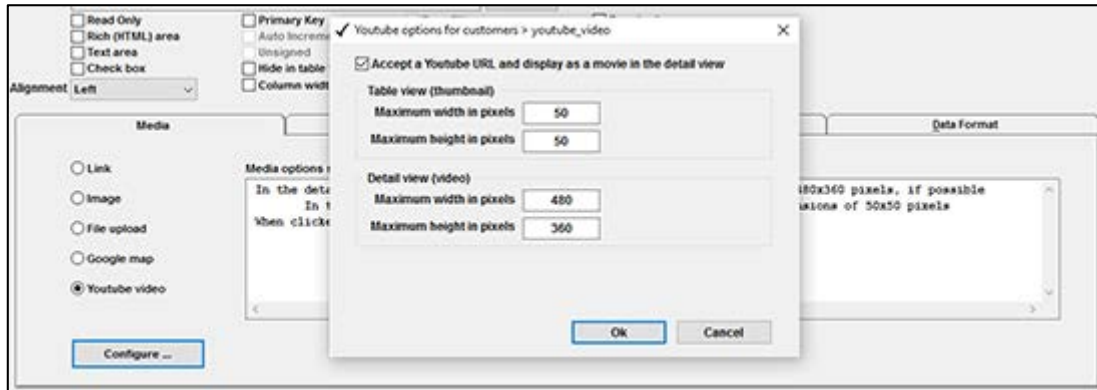


In the table view, the map is displayed as a thumbnail image.

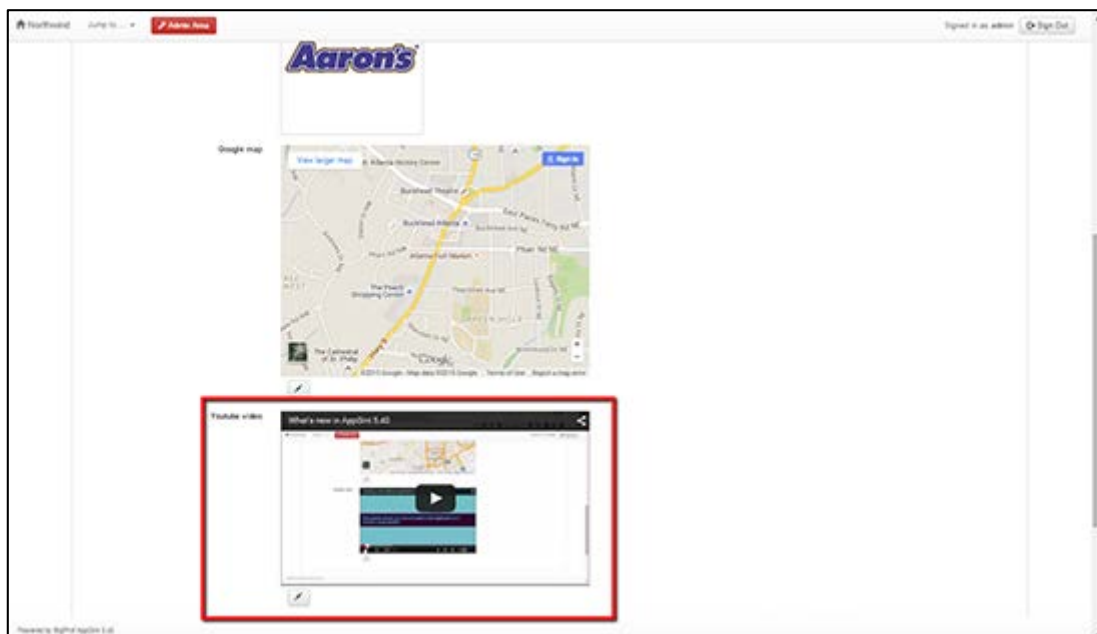
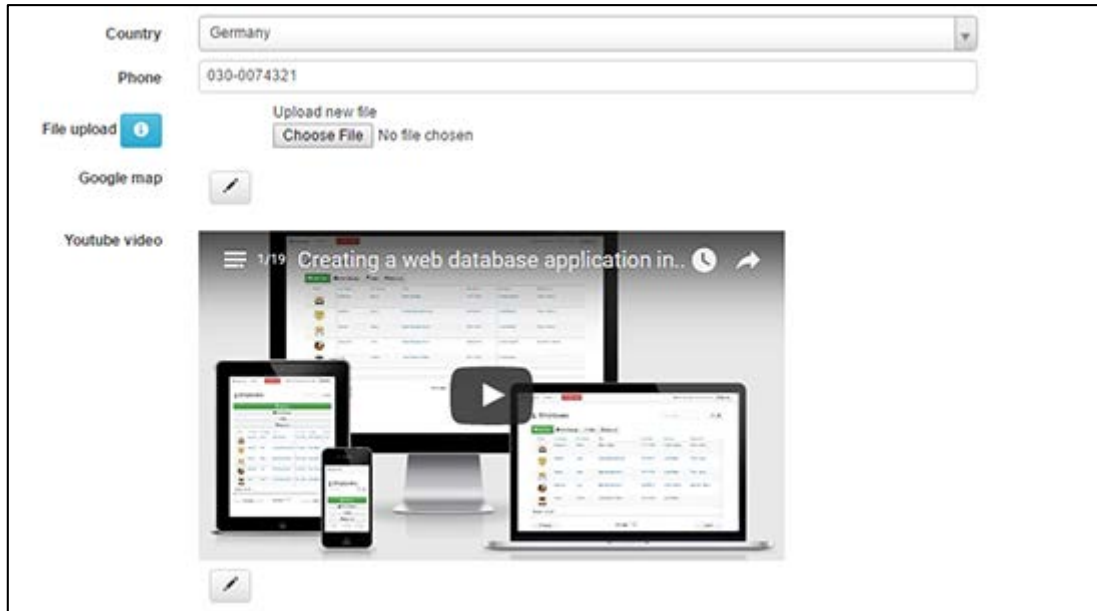


YouTube video

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.



Understanding lookup fields

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

ReportsTo

- ☐ Buchanan, Steven
- ☐ Callahan, Laura
- ☐ Davolio, Nancy
- ☐ Dodsworth, Anne
- ☒ Fuller, Andrew
- ☐ King, Robert
- ☐ Leverling, Janet
- ☐ Peacock, Margaret
- ☐ Suyama, Michael

'ReportsTo' is set as a lookup field shown as radio buttons rather than a drop-down menu.

Displaying lookup fields as an options list

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.

Media **Lookup field** Options list Data Format

Parent table
employees

Parent caption field part 1 Separator Parent caption field part 2
LastName . Firstname

☒ Show as radio buttons ☐ Inherit access permissions

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.

Related screen casts

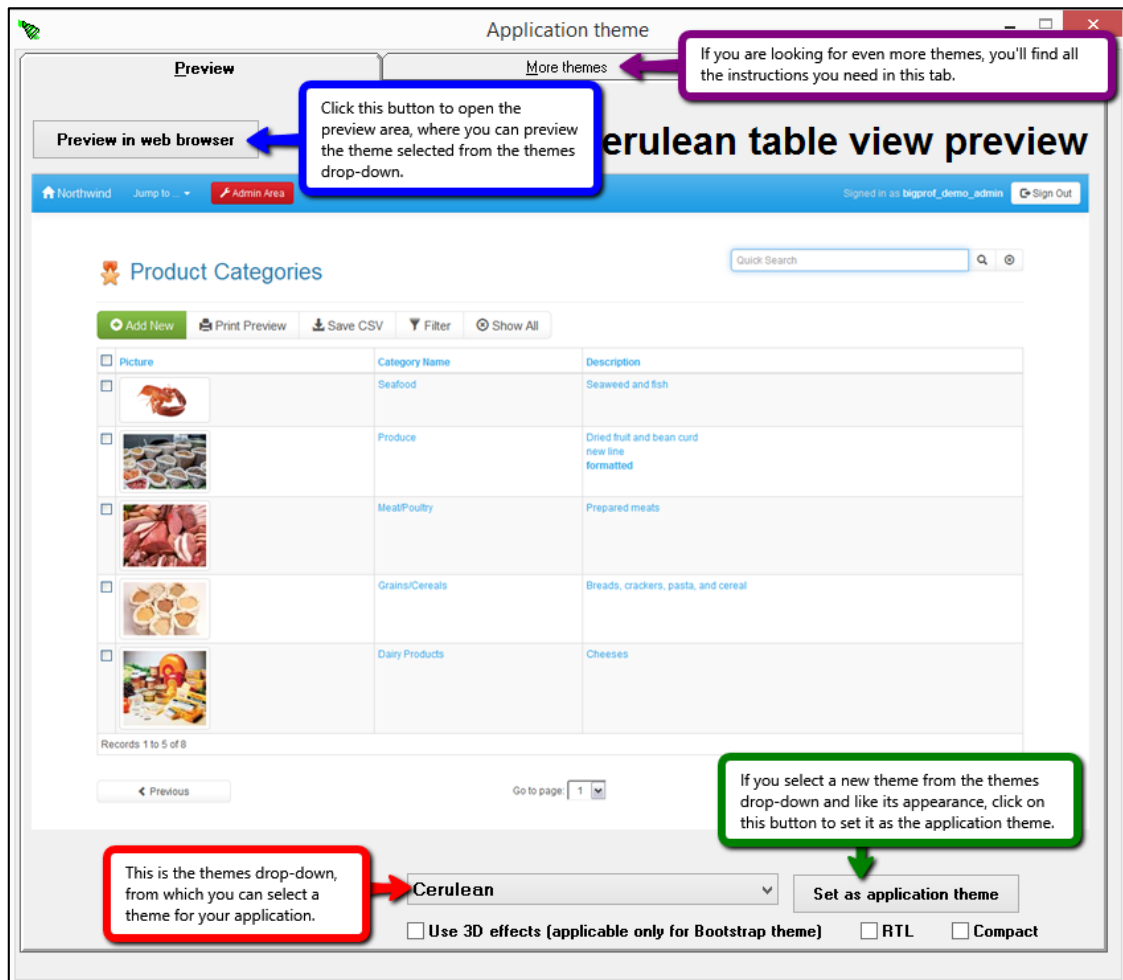
[AppGini lookup fields and master detail pages](#)

[Using auto-fill look-up fields to automatically populate fields from another table](#)

[Creating cascading drop downs with AppGini](#)

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.



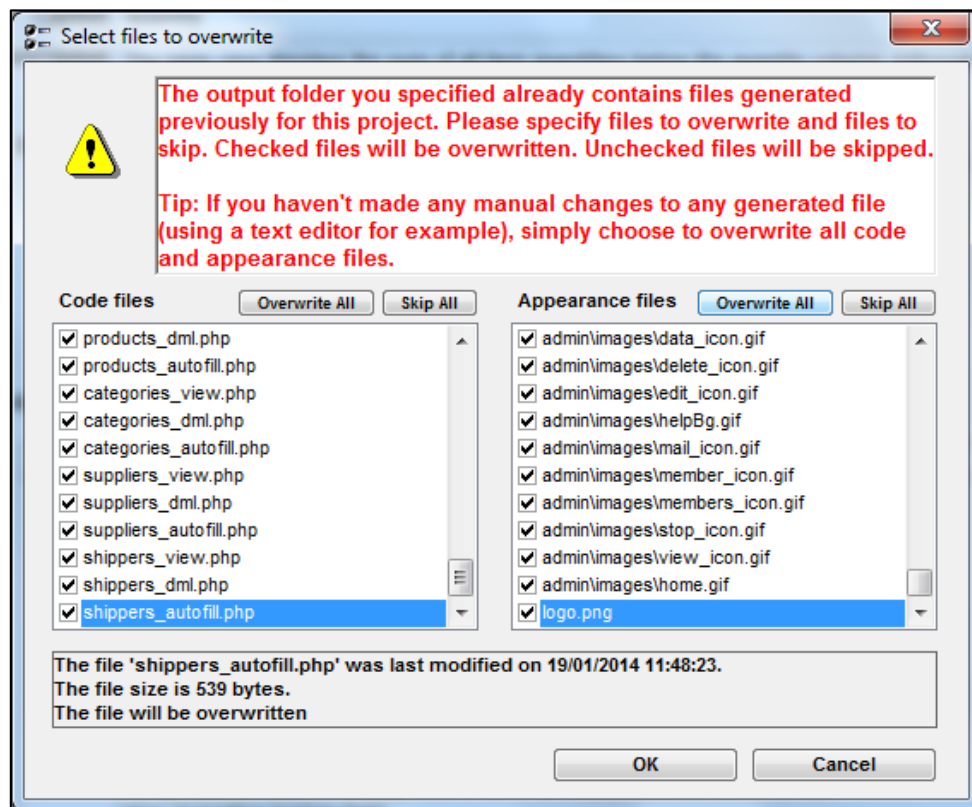
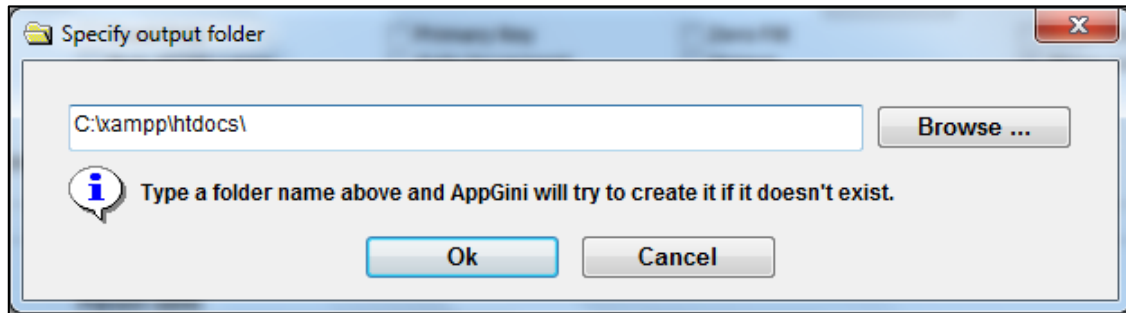
You can select a theme from the drop-down menu at the bottom. You can also click the **Customize theme** tab to edit the theme CSS and you can preview your changes instantly in the "Preview" tab .

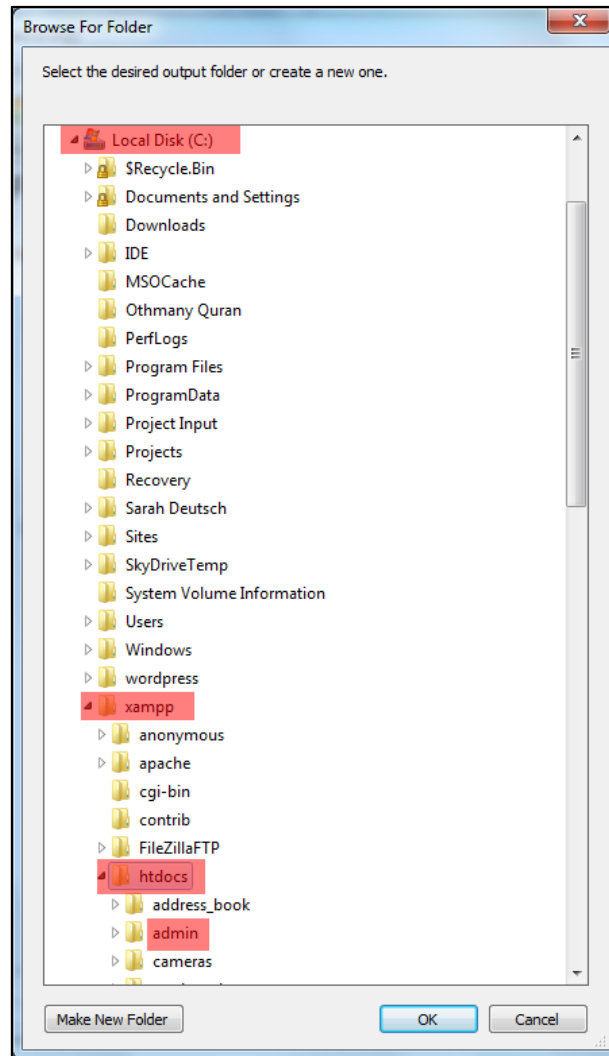
Tip: AppGini comes with several themes ready to use. You can add your own (or edit the ones that come with AppGini) under the sub-folder "add-ons\themes" inside the folder where AppGini is installed (usually "C:\Program Files\AppGini" or "C:\Program Files (x86)\AppGini")

For a quick reference on CSS style sheets, see [SitePoint CSS Reference](https://bigprof.com/appgini/).

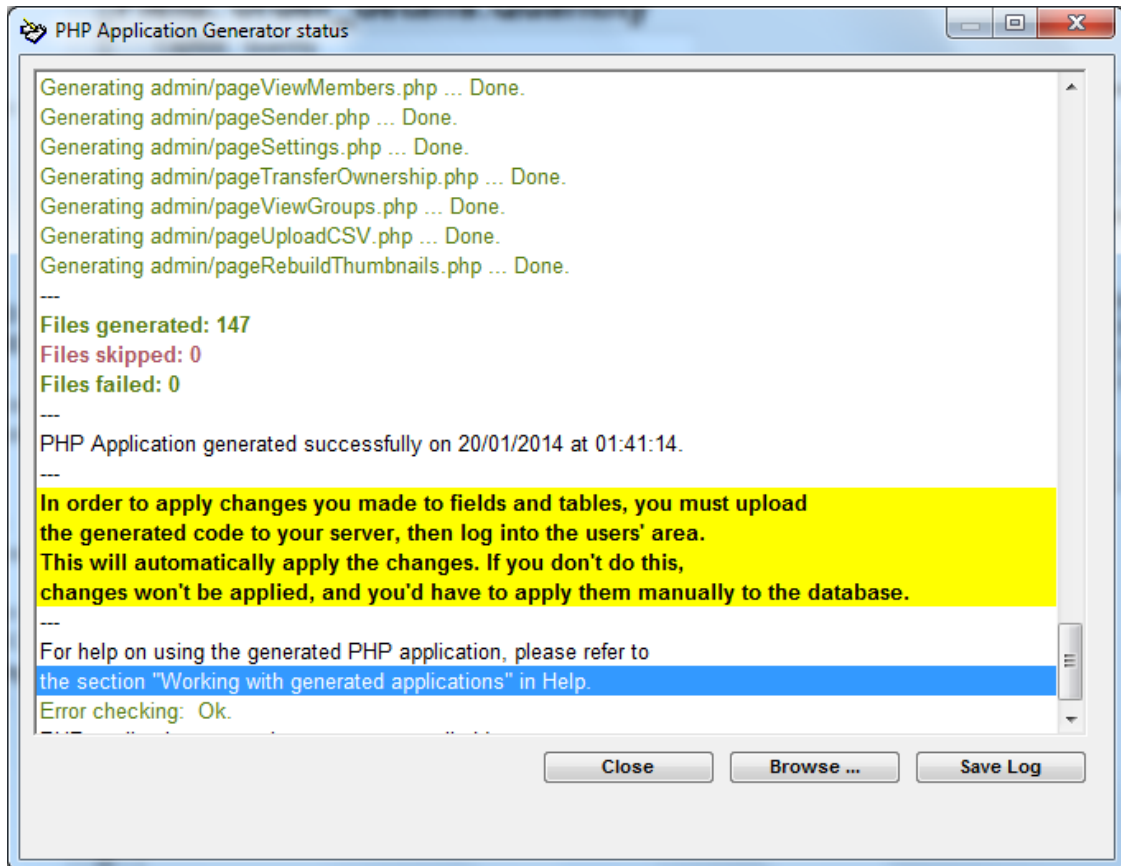
Generating the PHP application

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 PHP Code' icon. You'll be asked to select (or create) an output folder.





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

Keyboard shortcuts

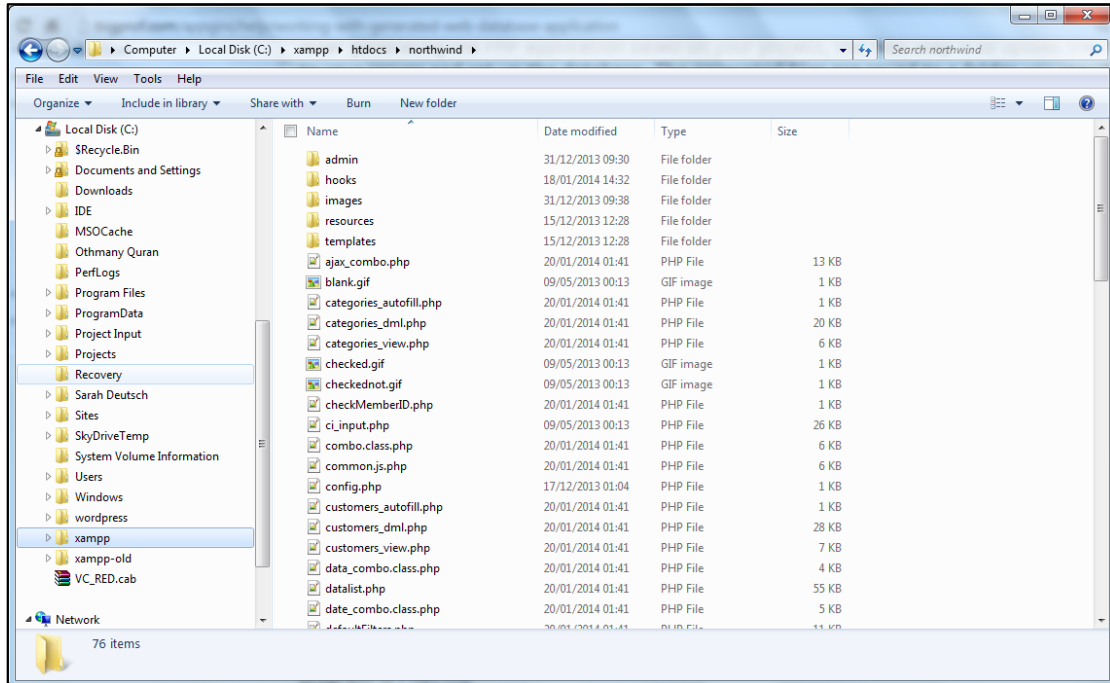
There are several keyboard shortcuts in AppGini that will help you work even faster with projects. Here is a list of them.

Ctrl+N	New project
Ctrl+O	Open an existing project
Ctrl+S	Save
Ctrl+Q	Quit
Ctrl+T	New Table
Ctrl+F	New Field
F5	Generate
F3	Properties
Shift+F3	Themes
Help	Shift+F1

Tip: Hold Ctrl while clicking the "Generate PHP Code" icon in AppGini to instantly generate your application using the most recent options you selected before (last output folder and file overwriting settings).

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.



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!

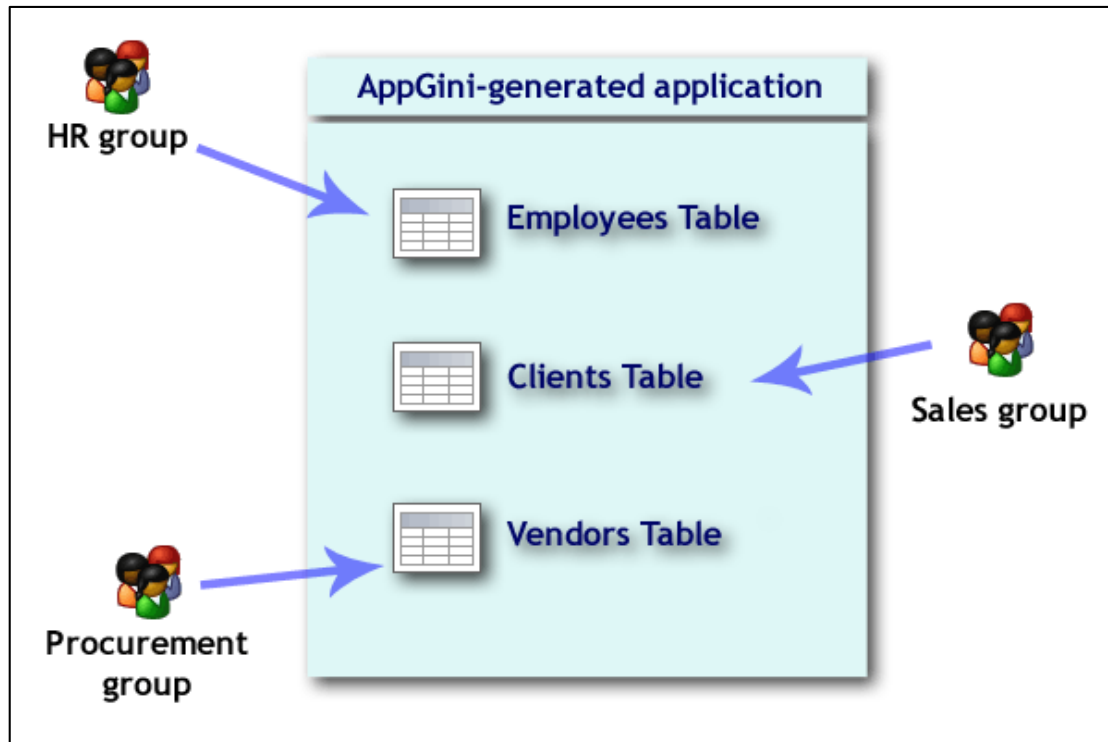
IMPORTANT Security notes

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 .

As of AppGini 4.0, the generated scripts implement 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

You may skip this section if you don't plan to modify the generated scripts.

For each table in your project, AppGini will generate 7 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. For example, you can edit the code for the insert() function to send you an email whenever a user adds a new 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

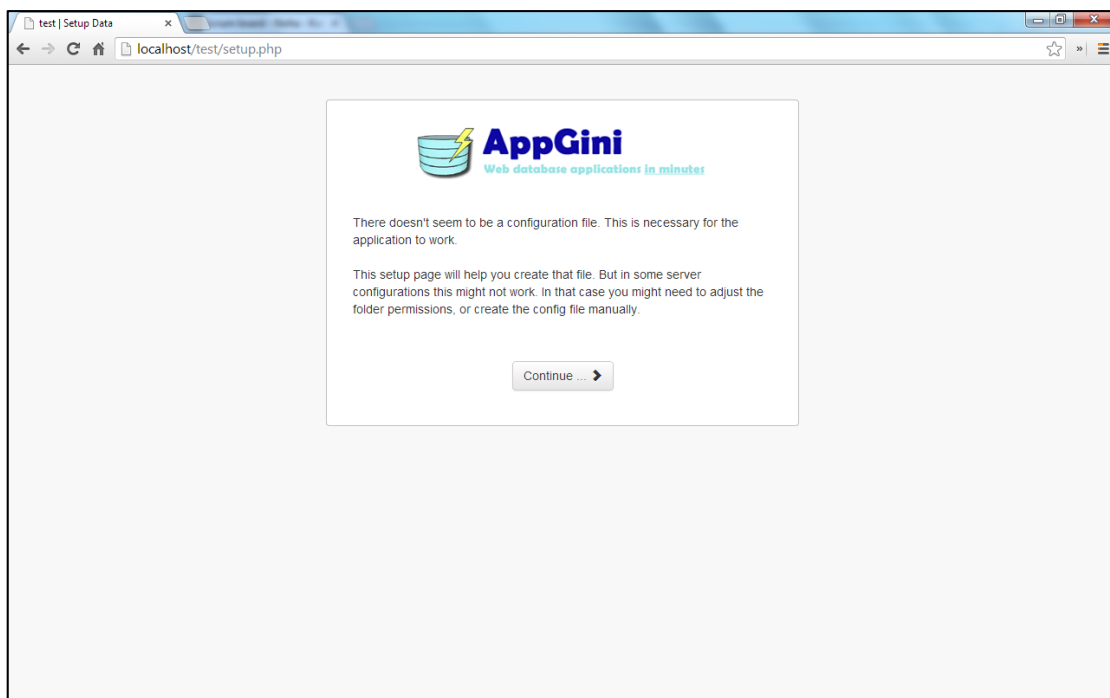
After uploading your PHP application files to a folder on your server, you can access it by pointing your browser to this URL :

http://www.yourserver.com/path_to_appgini_generated_app

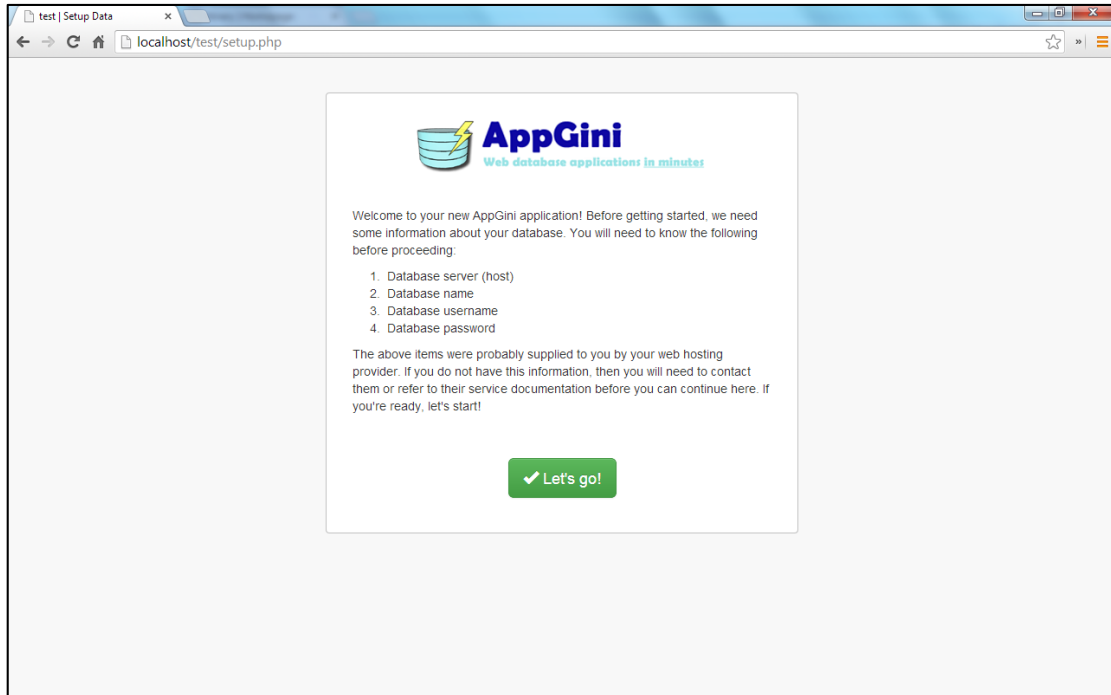
Replace 'www.yourserver.com' with your server name or IP, and 'path_to_appgini_generated_app' with the path to your application's folder .

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:



Clicking on the "setup page" link will open the setup script (or just wait for a few seconds and you'll be redirected automatically). This script will ask for the database login parameters, as shown in the screenshot below.



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 username and password you provided to AppGini have enough permissions to allow you to define the database and its tables .

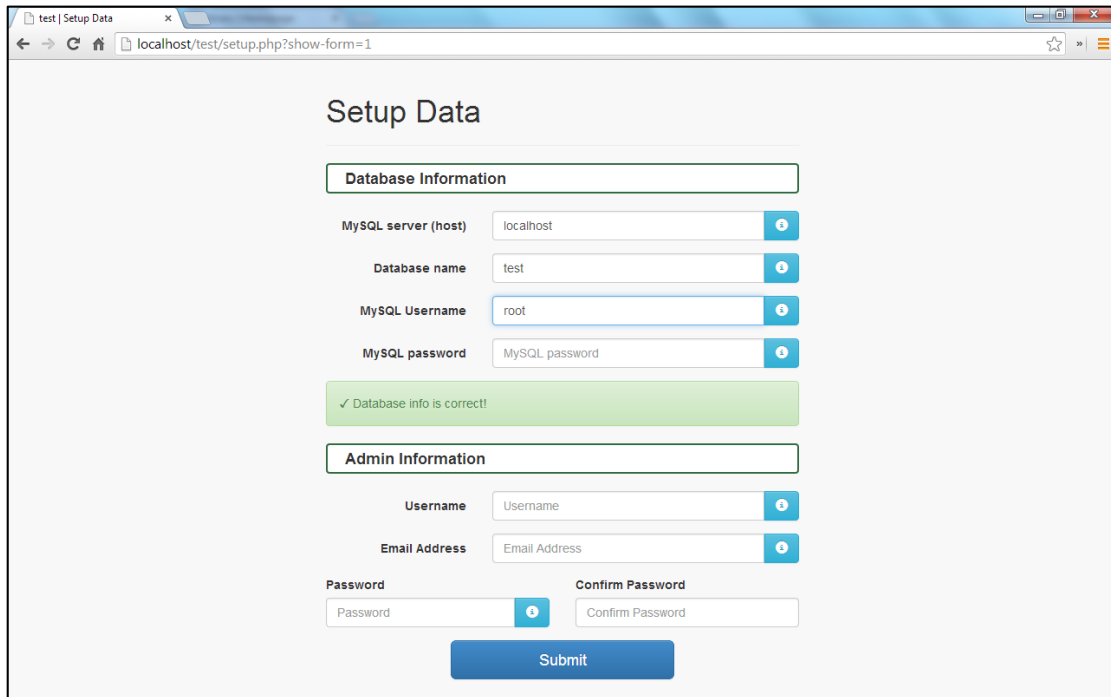
STEP 2 OF 3: Logging to the admin control panel

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

A screenshot of a web browser window showing the 'Setup Data' form. The browser's address bar displays 'localhost/test/setup.php?show-form=1'. The form is titled 'Setup Data' and is divided into two main sections: 'Database Information' and 'Admin Information'. The 'Database Information' section contains four input fields: 'MySQL server (host)' with the value 'localhost', 'Database name' with the value 'test', 'MySQL Username' with the value 'root', and 'MySQL password' with the placeholder 'MySQL password'. The 'Admin Information' section contains three input fields: 'Username' with the placeholder 'Username', 'Email Address' with the placeholder 'Email Address', and 'Password' with the placeholder 'Password'. There is also a 'Confirm Password' field with the placeholder 'Confirm Password'. A blue 'Submit' button is located at the bottom of the form.

APPGINI DOCUMENTATION

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



The screenshot shows a web browser window with the URL `localhost/test/setup.php?show-form=1`. The page is titled "Setup Data" and contains two main sections: "Database Information" and "Admin Information".

Database Information:

- MySQL server (host):
- Database name:
- MySQL Username:
- MySQL password:

A green message box indicates: "✓ Database info is correct!"

Admin Information:

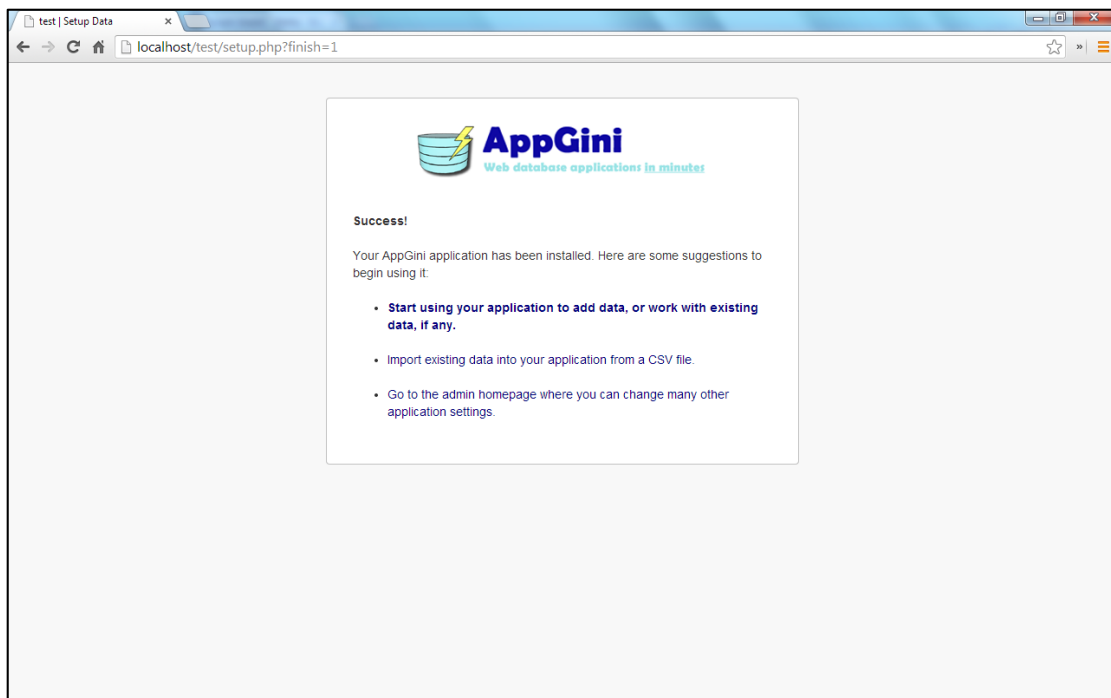
- Username:
- Email Address:
- Password:
- Confirm Password:

A blue "Submit" button is located at the bottom of the form.

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 .

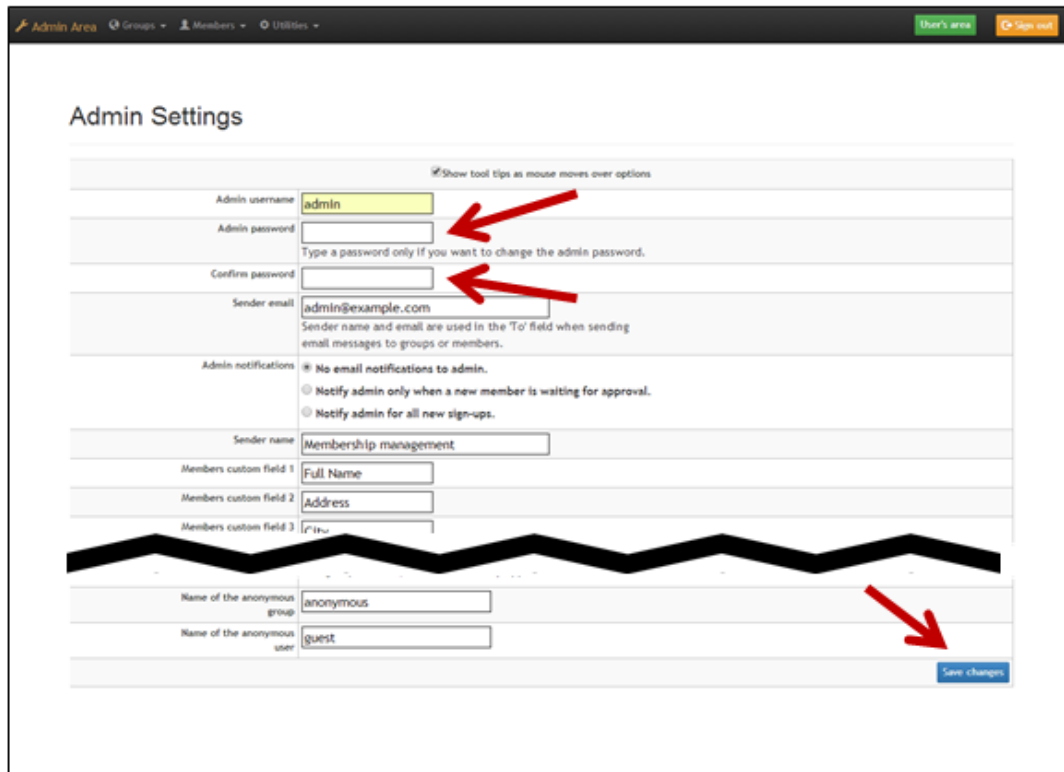
STEP 3 OF 3: Changing the admin password

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



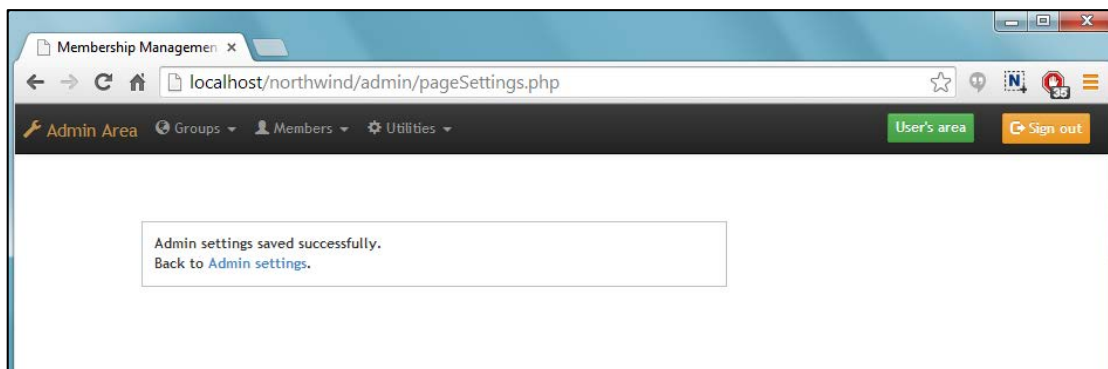
APPGINI DOCUMENTATION

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":



The screenshot shows the "Admin Settings" page. At the top, there is a navigation bar with "Admin Area", "Groups", "Members", and "Utilities". On the right, there are links for "User's area" and "Sign out". The main content area is titled "Admin Settings" and includes a checkbox for "Show tool tips as mouse moves over options". The form contains several fields: "Admin username" (set to "admin"), "Admin password" (with a red arrow pointing to it), "Confirm password" (with a red arrow pointing to it), "Sender email" (set to "admin@example.com"), "Admin notifications" (with radio buttons for "No email notifications to admin.", "Notify admin only when a new member is waiting for approval.", and "Notify admin for all new sign-ups."), "Sender name" (set to "Membership management"), "Members custom field 1" (set to "Full Name"), "Members custom field 2" (set to "Address"), "Members custom field 3" (set to "Phone"), "Name of the anonymous group" (set to "anonymous"), and "Name of the anonymous user" (set to "guest"). A red arrow points to the "Save changes" button at the bottom right.

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



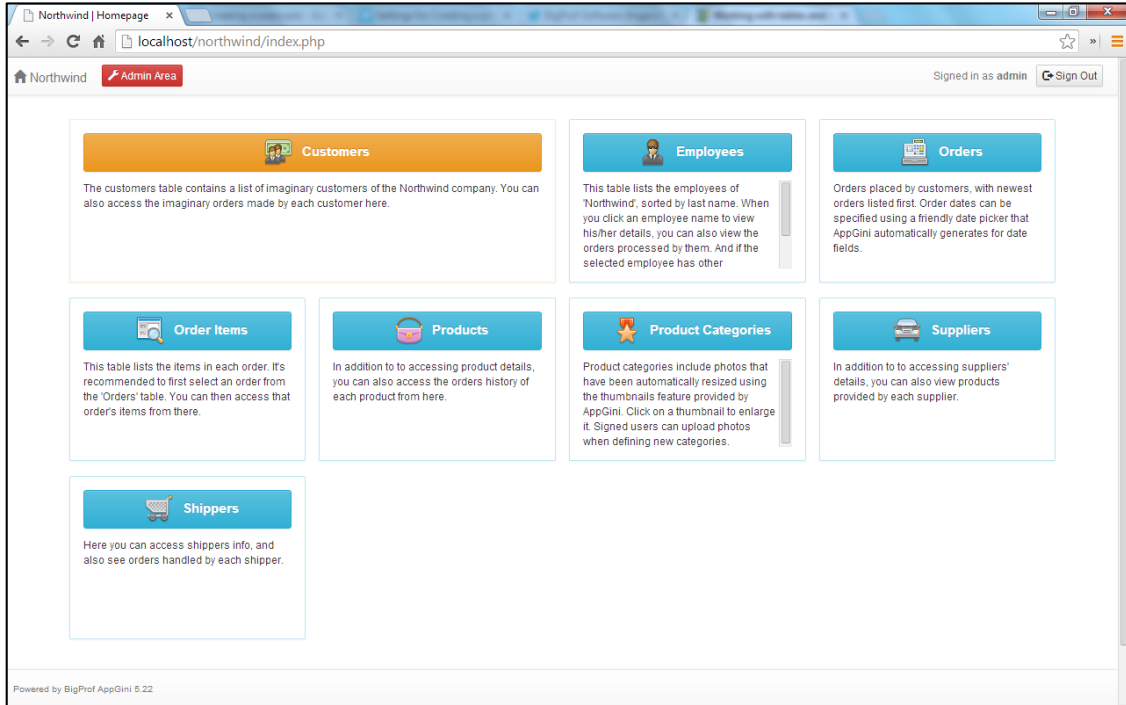
If you'd like to have a look on the generated table pages, Click "Sign Out", and then click the "Go to user's area" link from the following page .

For more information about using the admin area and managing users, please refer to [the admin interface](#) section.

Working with tables and records

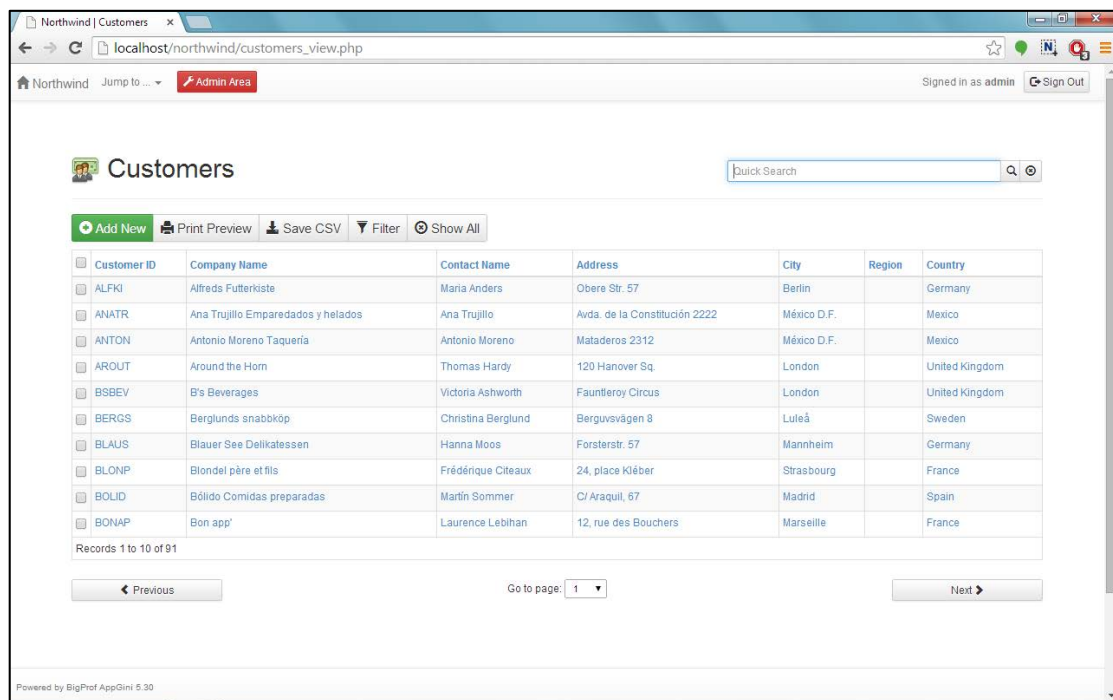
The application homepage

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.



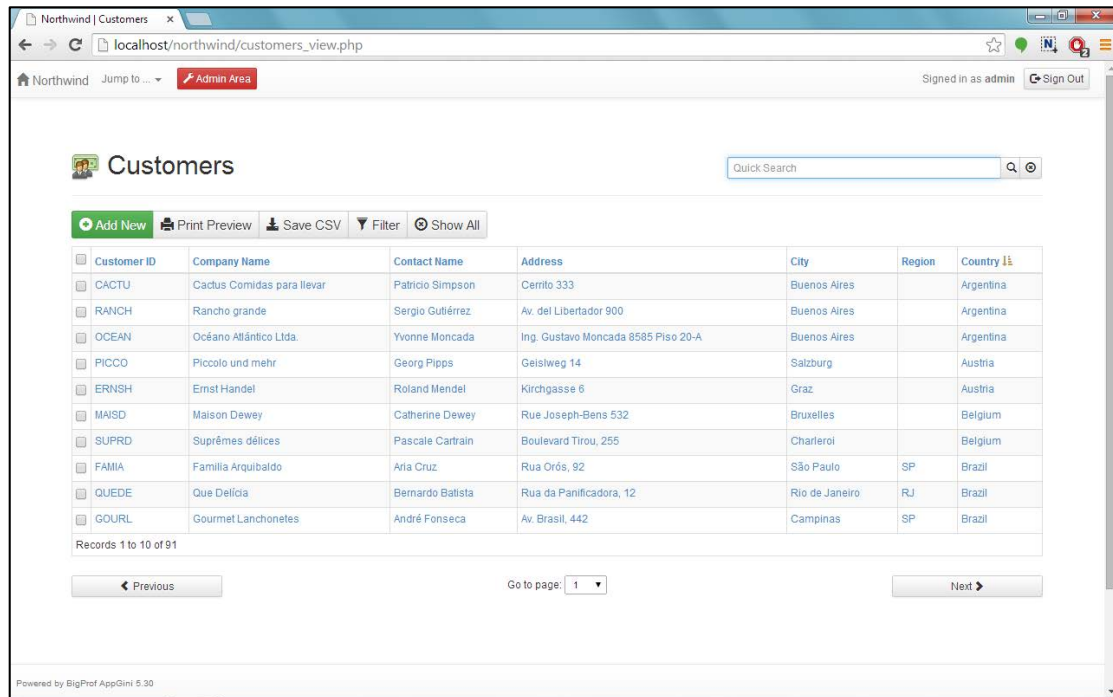
The Table View page

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.



APPGINI DOCUMENTATION

Column headers are what you specified in AppGini 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.

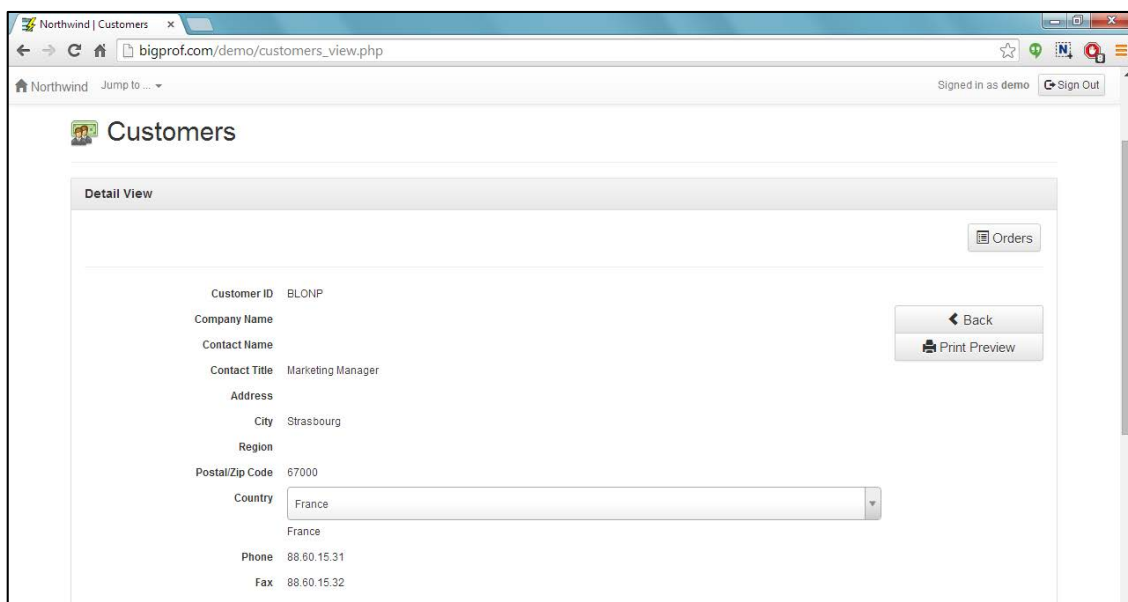


The screenshot shows a web browser window displaying the 'Customers' table in AppGini. The table is sorted by 'Country' in ascending order. The interface includes a search bar, action buttons (Add New, Print Preview, Save CSV, Filter, Show All), and pagination controls. The footer indicates it is powered by BigProf AppGini 5.30.

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

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) .

The Detail View (shown below) 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 the details not shown in the table view, editing, deleting, printing, and/or copying to a new record. The exact functions enabled depend on the permissions of the logged user.



The screenshot shows the 'Detail View' for a customer record in AppGini. The form displays the following information:

- Customer ID: BLONP
- Company Name
- Contact Name
- Contact Title: Marketing Manager
- Address
- City: Strasbourg
- Region
- Postal/Zip Code: 67000
- Country: France (selected from a dropdown menu)
- Phone: 88.60.15.31
- Fax: 88.60.15.32

Buttons for 'Orders', 'Back', and 'Print Preview' are visible on the right side of the form.

The screenshot shows the 'Customers' detail view in AppGini. The browser address bar shows 'bigprof.com/demo/customers_view.php?SelectedID=Drani'. The page title is 'Customers'. The 'Detail View' section contains the following fields:

- Customer ID: DRANI
- Company Name: Leipzig cuisine GmbH
- Contact Name: Stephan Shiffer
- Contact Title: Sales Agent
- Address: Schillerweg 36, Leipzig
- City: Saxony
- Region:
- Postal/Zip Code: 4155
- Country: Germany
- Phone: +49 341 24895743
- Fax: +49 341 24892546

On the right side, there are buttons: 'Orders', 'Save Changes' (green), 'Cancel' (orange), 'Print Preview' (grey), and 'Delete' (red). The footer indicates 'Powered by BigProf AppGini 5.30'.

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.

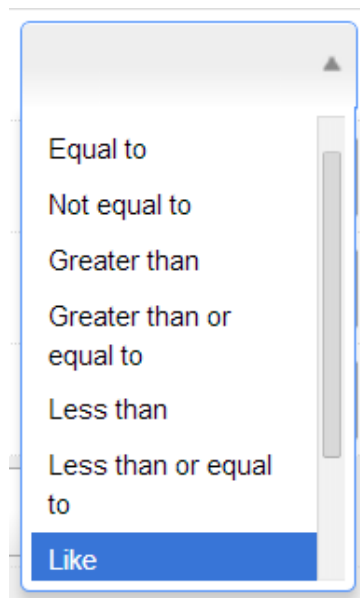
The screenshot shows the 'Customers Filters' page in AppGini. The page title is 'Customers Filters'. It contains the following sections:

- Filtering Section:** A table with columns 'Filtered field', 'Comparison Operator', and 'Comparison Value'. It contains four filter rows (Filter 01 to Filter 04) and a 'Filter 05' row. Each row has a dropdown for the field, a dropdown for the operator, and a text input for the value. There are also 'Add' and 'Remove' buttons for each filter.
- Order by Section:** A section for sorting records. It includes 'Order by' and 'Then by' dropdowns, each with a corresponding text input for the sort value.
- Records to display Section:** A section for selecting which records to display. It includes three radio buttons: 'Only your own records', 'All records owned by your group', and 'All records'.
- Action Buttons:** At the bottom, there are three buttons: 'Apply filters' (green), 'Save and apply filters' (grey), and 'Cancel' (orange).

The footer indicates 'Powered by BigProf AppGini 5.40'.

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 figure shows them all.



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

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 .

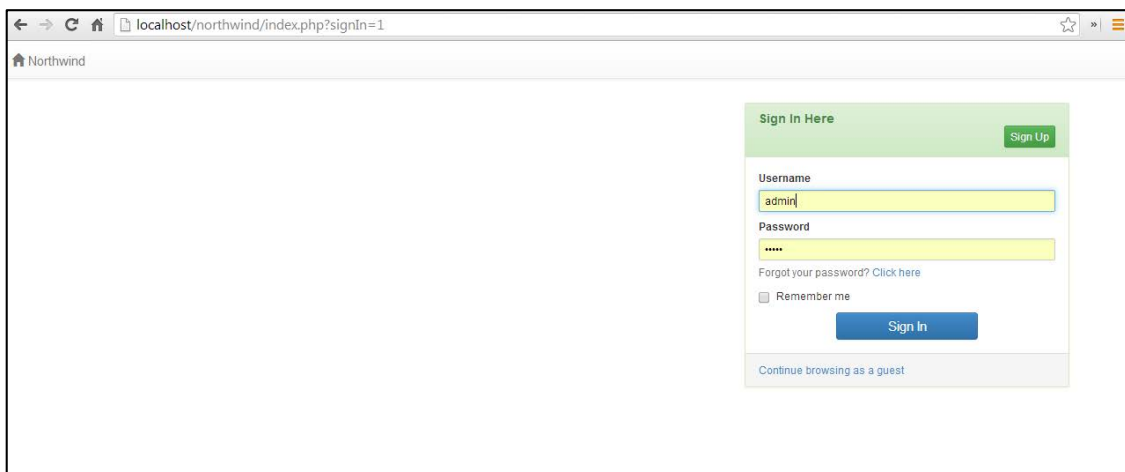
Accessing the admin interface

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 using this URL:

http://www.yourserver.com/path_to_AppGini_generated_app/admin

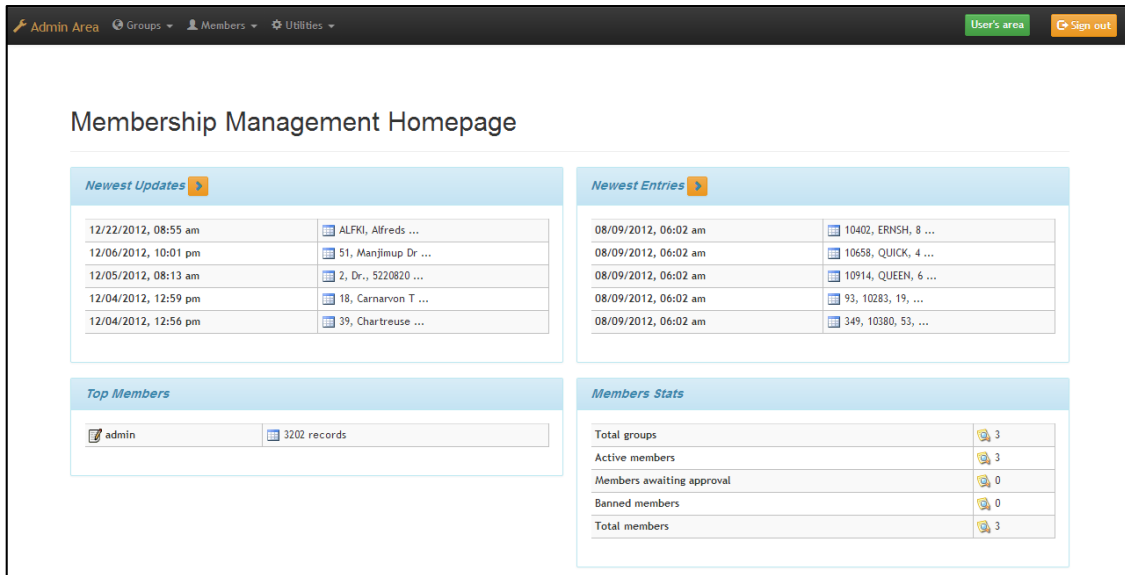
Of course, you should change the items in *italic* in the above URL to your actual server name and application path. The above URL should display the following screen.



The default username is admin, and the default password is admin. When you sign in for the first time, you should change them from the Admin Settings page.

The admin homepage

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.



Membership Management Homepage

Newest Updates

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 ...

Newest Entries

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, ...

Top Members

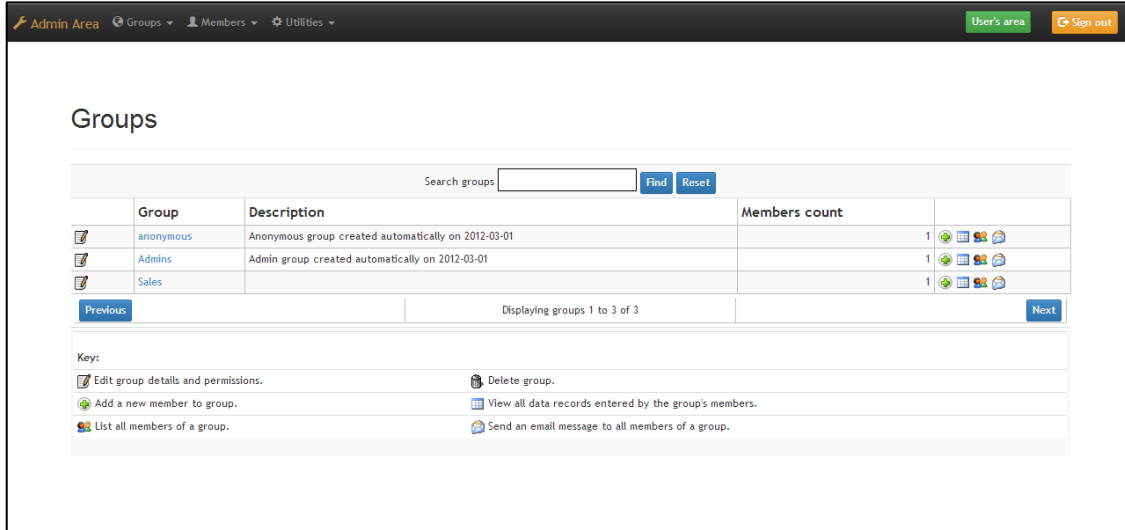
admin	3202 records
-------	--------------

Members Stats

Total groups	3
Active members	3
Members awaiting approval	0
Banned members	0
Total members	3

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.



Groups

Search groups [Find](#) [Reset](#)

Group	Description	Members count
anonymous	Anonymous group created automatically on 2012-03-01	1
Admins	Admin group created automatically on 2012-03-01	1
Sales		1

[Previous](#) [Next](#) Displaying groups 1 to 3 of 3

Key:

- Edit group details and permissions.
- Add a new member to group.
- List all members of a group.
- Delete group.
- View all data records entered by the group's members.
- Send an email message to all members of a group.

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.

Edit Group 'Admins'

☒ Show tool tips as mouse moves over options

Group name:
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

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.

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

To define a new group, click the 'Add Group' button on top of the admin homepage. 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.

Members

Search members in All fields Find Reset

Group Status Any

	Username	Group	Sign up date	Full Name	Address	City	State	Status	
	guest	anonymous	03/01/2012					Active	
	admin	Admins	03/01/2012					Active	
	demo	Sales	08/09/2012	Demo	Somewhere	Some City	Some State	Active	

Previous Next Displaying members 1 to 3 of 3

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.

Edit Member 'demo'

Member username

Password

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

Confirm password

Email

Group Sales

This user inherits the [permissions of his group.](#)
[Set special permissions for this user](#)

Approved? ☒

Banned? ☐

Full Name

Address

City

State

Comments

Save changes

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.

Data Records

Group: Member username: [Find](#) [Reset](#)

Show records from:

Sort records by:

Username	Group	Table	Created	Modified	Data
admin	Admins	orders	08/09/2012, 06:02 am	08/09/2012, 06:02 am	10832, LAMAI, 2, 1996-02-14, 1996-03-13, ...
admin	Admins	order_details	08/09/2012, 06:02 am	08/09/2012, 06:02 am	1151, 10686, 17, 39.00, 30, 0.20 ...
admin	Admins	orders	08/09/2012, 06:02 am	08/09/2012, 06:02 am	10674, ISLAT, 4, 1995-10-19, 1995-11-16, ...
admin	Admins	order_details	08/09/2012, 06:02 am	08/09/2012, 06:02 am	1689, 10893, 36, 19.00, 20, 0.00 ...
admin	Admins	order_details	08/09/2012, 06:02 am	08/09/2012, 06:02 am	993, 10623, 24, 4.50, 3, 0.00 ...
admin	Admins	orders	08/09/2012, 06:02 am	08/09/2012, 06:02 am	10516, HUNGO, 2, 1995-05-25, 1995-06-22, ...
admin	Admins	order_details	08/09/2012, 06:02 am	08/09/2012, 06:02 am	1531, 10833, 53, 32.80, 9, 0.10 ...
admin	Admins	orders	08/09/2012, 06:02 am	08/09/2012, 06:02 am	11054, CACTU, 8, 1996-05-28, 1996-06-25, ...
admin	Admins	orders	08/09/2012, 06:02 am	08/09/2012, 06:02 am	10358, LAMAI, 5, 1994-12-21, 1995-01-18, ...
admin	Admins	order_details	08/09/2012, 06:02 am	08/09/2012, 06:02 am	2069, 11052, 61, 28.50, 10, 0.20 ...

[Previous](#) Displaying records 1 to 10 of 3202 [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:

Owner member:

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.

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

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

Table:

Record data [Print](#)

Field name	Value
OrderID	10832
CustomerID	LAMAI
EmployeeID	2
OrderDate	1996-02-14
RequiredDate	1996-03-13
ShippedDate	1996-02-19
ShipVia	2
Freight	43.26
ShipName	LAMAI
ShipAddress	LAMAI
ShipCity	LAMAI
ShipRegion	LAMAI
ShipPostalCode	LAMAI
ShipCountry	LAMAI

[Save changes](#)

http://localhost/northwind/admin/pageEditOwnership.php?redO=888 Mon Jan 20 20:14 08:00:44 GMT+0200 (Egyp Standard Time)

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.

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](#)

Hooks

Hooks were added to AppGini as of version 4.50. Older versions don't support this feature.

AppGini Hooks is a means of advanced customization of AppGini-generated code. Prior to AppGini 4.50, code customization was a painful process, since all customizations you had manually made needed to be re-applied if you regenerated the code later. Even if using a CVS system to manage code changes, it was still a time-consuming task to compare code versions and re-apply customizations.

Hooks work by intercepting users' actions (inserts, deletes, edits, selection of records, ... etc), and controlling what happens on each action.

Using hooks, your code customizations are separate from the generated code. This way, they don't get overwritten on code regeneration and your project is ready for use directly after code generation without any further modifications.

How does it work?

To use hooks, you should place your code modifications in the "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. AppGini code 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.

The generated "hooks" folder

The hooks folder is where all your custom-defined code should be placed. AppGini generates the hook files into this folder only if they don't exist. AppGini doesn't overwrite these files later. So, your customized code is retained safely no matter how many times you regenerate your project code.

This folder contains the following files:

- **__global.php:** 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.
- **tablename.php:** For each table in your project, a hook file named the same as the table name is created. 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.
- **index.html:** This file should not be edited. It just redirects visitors who try to access the hooks folder to the main page .
- **links-home.php:** You can add custom links in the home page of your application by appending them to this file. The format for each link is:

```
$homeLinks[] = array(
    'url' => 'path/to/link',
    'title' => 'Link title',
    'description' => 'Link text',
    'groups' => array('group1', 'group2'),
    'grid_column_classes' => '',
    'panel_classes' => '',
    'link_classes' => '',
    'icon' => 'path/to/icon'
);
```

- 'groups' defines the groups allowed to see this link. Use '*' if you want to show the link to all groups.
 - 'grid_column_classes' (optional) lists CSS classes to apply to the link block. See: getbootstrap.com/css/#grid.
 - 'panel_classes' (optional) lists CSS classes to apply to the panel. See: getbootstrap.com/components/#panels.
 - 'link_classes' (optional) lists CSS classes to apply to link. See: getbootstrap.com/css/#buttons.
 - 'icon' is the path to an optional icon to use with the link.
- **links-navmenu.php:** You can add custom links to the navigation menu ("Jump to" menu) of your application by appending them to this file. The format for each link is:

```
$navLinks[] = array(
    'url' => 'path/to/link',
    'title' => 'Link title',
    'groups' => array('group1', 'group2'),
    'icon' => 'path/to/icon'
);
```

- 'groups' defines the groups allowed to see this link. Use '*' if you want to show the link to all groups.
- 'icon' is the path to an optional icon to use with the link.

Global hooks

Hooks were added to AppGini as of version 4.50. Older versions don't support this feature.

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. The following hook functions are defined in this file:

- `login_ok()`
- `login_failed()`
- `member_activity()`

`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 used but is reserved for future uses.

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
    if(!$fp=@fopen($logFile, 'a')) return '';
    fwrite($fp, "$date,$time,$username,$ip\n");
    fclose($fp);

    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($attempt, &$args){
}
```

Parameters:

- **\$attempt** is an associative array containing details of the failed login attempt, as follows:
 - \$attempt['username']: the username used during the failed login attempt.
 - \$attempt['password']: the password used during the failed login attempt.
 - \$attempt['IP']: the IP of the user who tried to log in.
- **\$args** is currently not used but is reserved for future uses.

Return value:

None.

Example:

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

```
function login_failed($attempt, &$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;
    }
}
```

Parameters:

- **\$memberInfo** is an array containing details of the member who signed in. Please refer to [memberInfo](#) for more details.
- **\$activity** A string that contains one of the following values:
 - 'pending': Means the member signed up through the signup form and awaits admin approval.
 - 'automatic': Means the member signed up through the signup form and was approved automatically.
- **\$args** is currently not used but is reserved for future uses.

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;
    }
}
```

Table-specific hooks

Hooks were added to AppGini as of version 4.50. Older versions don't support this feature.

For each table in your project, a hook file named the same as the table name is created. 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 if so 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 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 .

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 .

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** is currently not used but is reserved for future uses.

Return value:

A boolean TRUE to perform the insert operation, or FALSE to cancel it .

Example:

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;

}
```

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:

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, $memberInfo, &$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";
    }

    @mail(
        // mail recipient
        "employee@company.com",

        // subject
        "A new record needs your attention",

        // message
        "The following new record was submitted by {$memberInfo['username']}: \n\n".
            $messageData,

        // sender address
        "From: web@company.com"
    );

    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, $memberInfo, &$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** is currently not used but is reserved for future uses.

Return value:

True to perform the update operation or false to cancel it .

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, $memberInfo, &$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:

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($selectedID, &$skipChecks, $memberInfo, &$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, $memberInfo, &$amp;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, $memberInfo, &$amp;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, $memberInfo, &$amp;html, &$amp;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:

In this (long) example, we'll customize the detail view of the Orders table of the [Northwind demo](#) so that if an order is selected, the order items are listed below it. Such view is typically known as a master/detail view. The screen shot below shows the detail view of the Orders table before adding the example code.

 Orders

Detail View

Order Items

✓ Save Changes

← Back

Print Preview

Delete

Order ID

11077

Customer

Rattlesnake Canyon Grocery

Employee

Davolio, Nancy

Order Date

June

5

1996

Required Date

July

3

1996

Shipped Date

Ship Via

United Package

Freight

8.53

Ship Name

Rattlesnake Canyon Grocery

Ship Address

2817 Milton Dr.

Ship City

Albuquerque

Ship Region

NM

Ship Postal Code

87110

Ship Country

USA

```

function orders_dv($selectedID, $memberInfo, &$amp;html, &$amp;sargs){

    // if an order is selected, display its details.
    if($selectedID){

        $html.='<h3>Order Details</h3>';
        $id=makeSafe($selectedID);

        // get order items
        $res=mysql(
            "select  ProductName, order_details.UnitPrice, Quantity ".
            "from order_details left join products ".
            "on order_details.ProductID=products.ProductID ".
            "where OrderID='$id'"
        , $eo);

        if(mysql_num_rows($res)){

            // list order items inside a table
            $html.='<table>';

            // column titles
            $html.='<tr>'.
                '<td>Item</td>'.
                '<td>Unit price</td>'.
                '<td>Quantity</td>'.
                '<td>Price</td>'.
                '</tr>';

            // iterate through order items, calculating item prices and order subtotal
            while($row=mysql_fetch_row($res)){
                $price = $row[1] * $row[2];
                $subtotal+=$price;

                $html.="<tr>".
                    "<td>$row[0]</td>".
                    "<td>$row[1]</td>".
                    "<td>$row[2]</td>".
                    "<td>$price</td>".
                    "</tr>";
            }

            // order subtotal
            $html.="<tr>".
                '<td colspan="3">Subtotal</td>'.
                "<td>$subtotal</td>".
                "</tr>";

            $html.='</table>';
        }
    }
}

```

The screen shot below shows the detail view of the Orders table after adding the example code into the orders_dv() hook function. Additional formatting was applied in this screenshot for better visibility but I omitted it from the above example for simplicity.

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.

APPGINI DOCUMENTATION

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.

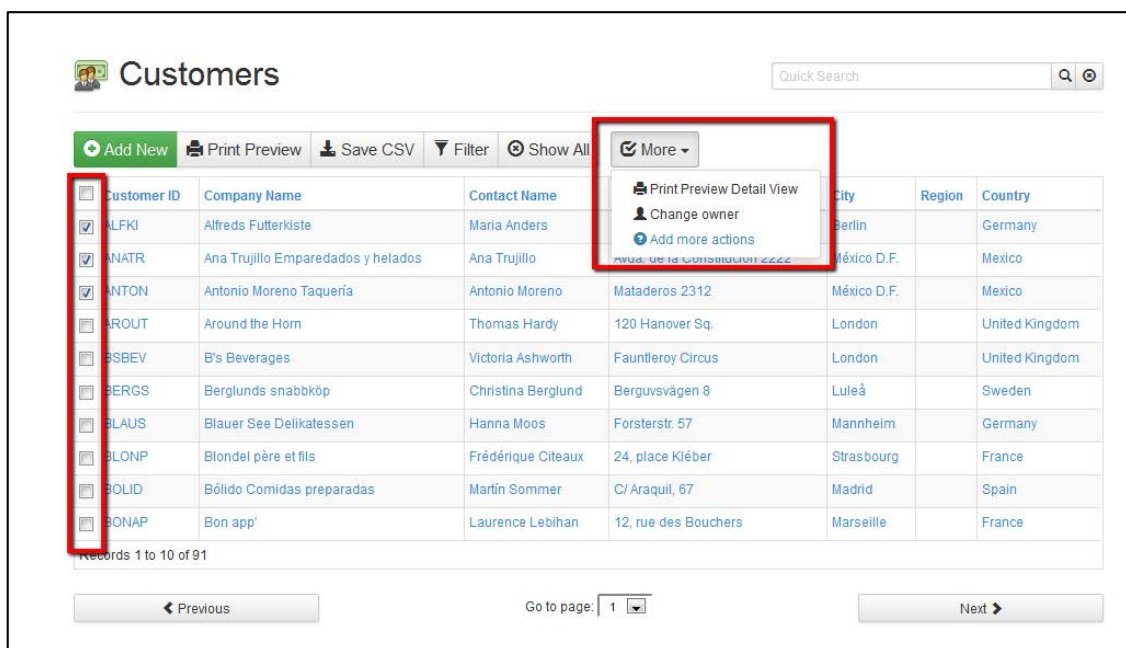
```
function tablename_csv($query, $memberInfo, $args){

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

    return $query;
}
```

Adding custom "batch actions" that apply to multiple records

AppGini 5.30 introduced a new feature: batch actions. 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.



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 .

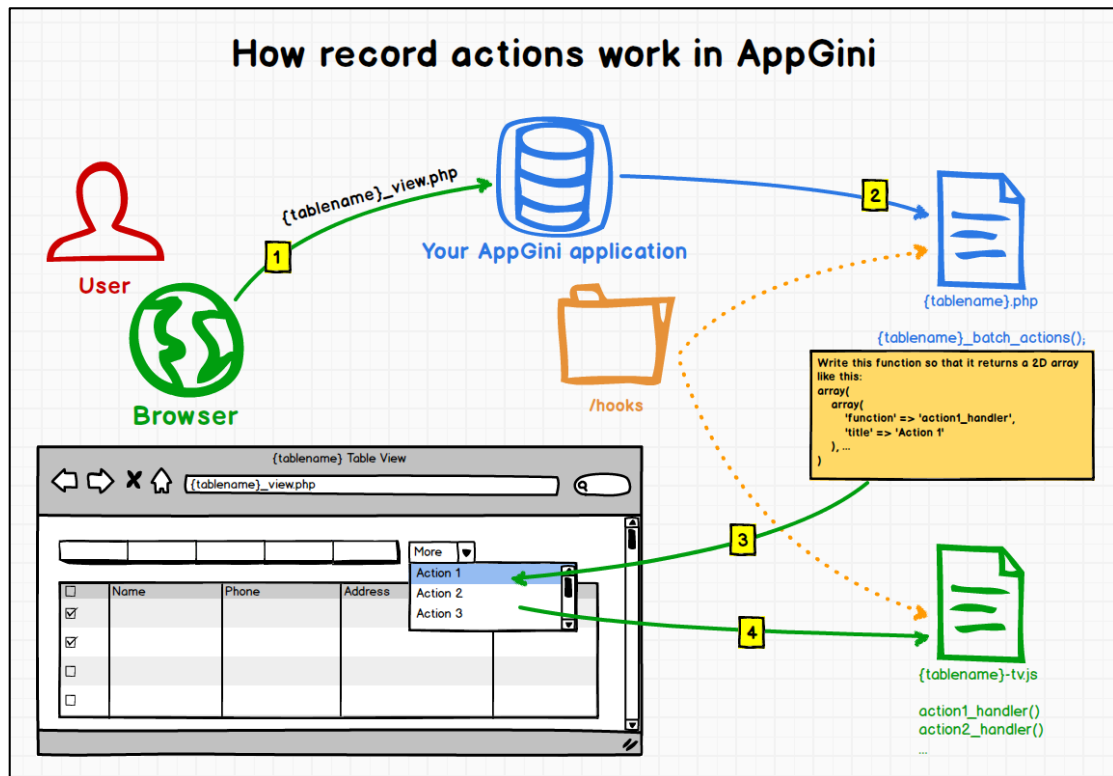
You can add more actions!

For power users, we've added a new hook function, {tablename}_batch_actions(). In this hook function, you can define new batch actions. You just define them by name in the function, but you add their details and functionality in another place that we'll come to in a moment. The 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 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.



So, here is what happens:

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.

Let's write an example batch action

So, let's assume that for the customers table in the screenshot above, we want to add a batch action for printing mail labels. That is, the user selects one or more customers, opens the "More" menu, and chooses "Print mail labels" (this is the action we'll write in this example). This would open a new page displaying printable mail labels for the selected customers .

So, the first step is to add the action into the `customers_batch_actions` hook. To do so, we'll open the `customers.php` file. This file should be in the "hooks" folder. If we scroll down the file, we should find our hook function:

```
function customers_batch_actions(&$args){

    return array();

}
```

Note: If you generated your application with a version of AppGini older than 5.30 before, you won't find the above function in the `customers.php` file. If so, just insert it at the end of the file.

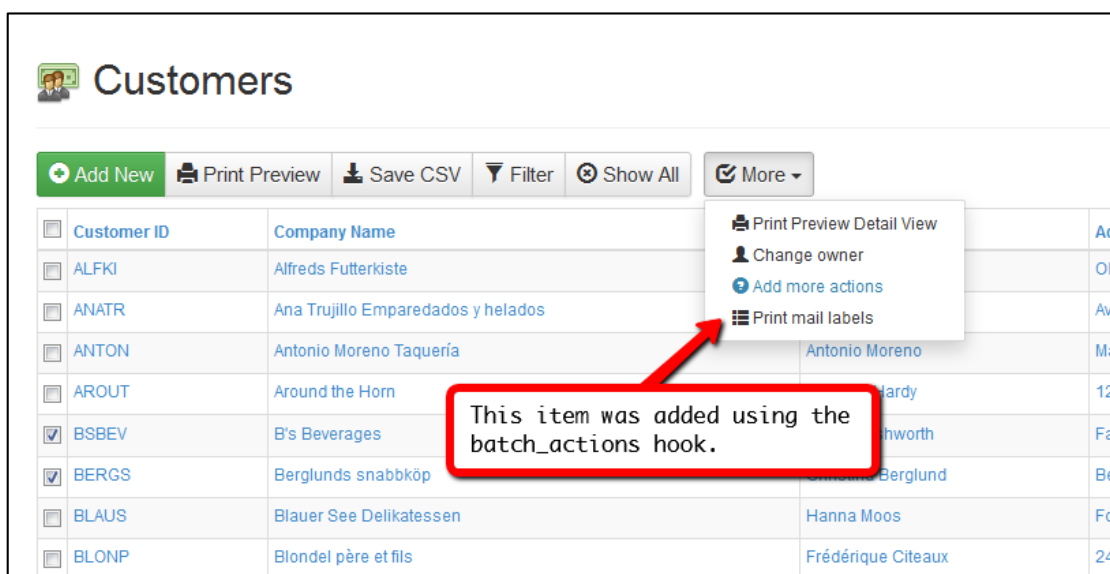
The function above is empty (we call this a skeleton function). We need to add our action to it. So, let's modify it to read:

```
function customers_batch_actions(&$args){

    return array(
        array(
            '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 have a look on the "More" menu after adding the code above.



We 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 see 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.

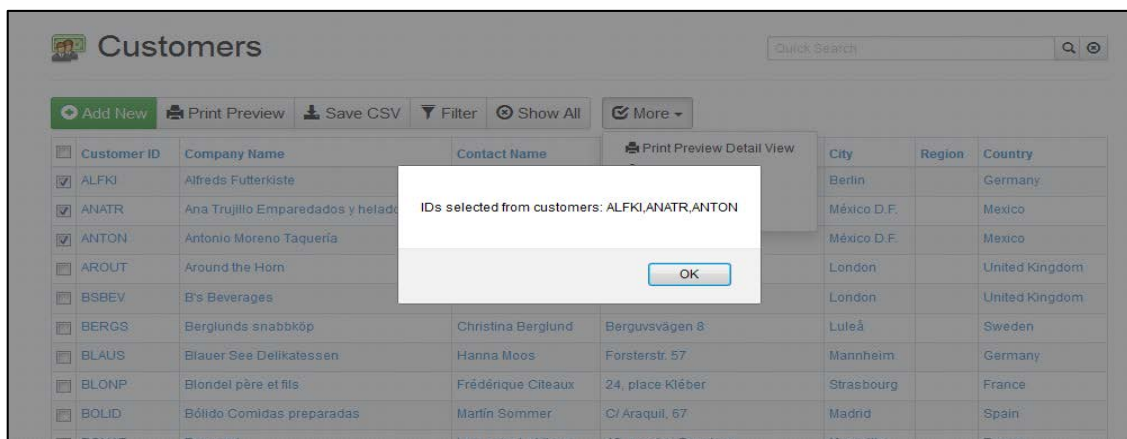
APPGINI DOCUMENTATION

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, just create it there .. the format is {tablename}-tv.js. If AppGini finds this file in the hooks folder, it tells the browser to load it 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
    */
    var url = 'mail-labels.php?table=' + table_name;
    for(var i = 0; i < ids.length; i++){
        url = url + '&'
            + encodeURIComponent('ids[' + i + ']') + '='
            + 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
*/
$curr_dir = dirname(__FILE__);
include("{ $curr_dir }/defaultLang.php");
include("{ $curr_dir }/language.php");
include("{ $curr_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 */
$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 <i>C/O Maria Anders</i> Obere Str. 57 Berlin 12209 Germany
Ana Trujillo Emparedados y helados <i>C/O Ana Trujillo</i> Avda. de la Constitución 2222 México D.F. 05021 Mexico
Antonio Moreno Taquería <i>C/O Antonio Moreno</i> Mataderos 2312 México D.F. 05023 Mexico
Berglunds snabbköp <i>C/O Christina Berglund</i> Bergmussvägen 8

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. Happy coding!

DataList object

Hooks were added to AppGini as of version 4.50. Older versions don't support this feature.

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

Property	Description
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 <i>QueryFieldsTV</i> property explained below.
ColWidth	An array that specifies the width of each column in the table view. If the <i>ShowTableHeader</i> property (explained below) is set to 1, the <i>ColWidth</i> property is overridden by the width values specified in the table view template file (templates/ <i>tablename_templateTV.html</i>).
DefaultSortDirection	A string that can be set to 'asc' or 'desc'. Please see the <i>DefaultSortField</i> 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 <i>QueryFieldsTV</i> property explained below. Alternatively, it can be set to a string specifying an explicit field name or MySQL expression to use for default sorting.
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.

Property	Description
HighlightColor	A string that specifies the color to use for highlighting table view records when the mouse passes over them. Uses HTML color syntax, for example '#FFF0C2'.
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 <i>ColNumber</i> 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	<p>A number that specifies how to display the quick search box. It can take any of the following values:</p> <ul style="list-style-type: none"> 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. <p>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.</p>
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.

Property	Description
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 <i>ColCaption</i> 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 <i>templates/tablename_templateTV.html</i> 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.

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

Hooks were added to AppGini as of version 4.50. Older versions don't support this feature.

\$memberInfo is an associative array containing logged member's info, as follows:

- **\$memberInfo['username']**: the member username.
- **\$memberInfo['groupID']**: the numeric ID of the member's group.
- **\$memberInfo['group']**: the name of the member's group.
- **\$memberInfo['admin']**: true for admin member, false for others.
- **\$memberInfo['email']**: the email address of the member.
- **\$memberInfo['custom'][0]**: the value of the first custom field for the member.
- **\$memberInfo['custom'][1]**: the value of the second custom field for the member.
- **\$memberInfo['custom'][2]**: the value of the third custom field for the member.
- **\$memberInfo['custom'][3]**: the value of the fourth custom field for the member.
- **\$memberInfo['IP']**: the IP address from where the member is currently logged.

The `$memberInfo` array is passed to the following hook functions:

- [login_ok\(\)](#)
- [member_activity\(\)](#)
- [tablename_init\(\)](#)
- [tablename_header\(\)](#)
- [tablename_footer\(\)](#)
- [tablename_before_insert\(\)](#)
- [tablename_after_insert\(\)](#)
- [tablename_before_update\(\)](#)
- [tablename_after_update\(\)](#)
- [tablename_before_delete\(\)](#)
- [tablename_after_delete\(\)](#)
- [tablename_dv\(\)](#)
- [tablename_csv\(\)](#)

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.

```
document.observe('dom:loaded', function() {
    $('score').observe('change', function() {
        if(isNaN($F('score')) || $F('score') > 100 || $F('score') < 0){
            alert('Score must be between 0 and 100!');
            $('score').focus();
        }
    });
});
```

Line 1 in the code above makes sure this code won't be executed until the page content is loaded to avoid triggering an error on some browsers. AppGini automatically loads the [Prototype javascript framework](#). So, the above example makes use of that framework .

tablename-tv.js: modifying the behavior of the table view through javascript

As of AppGini 5.30, 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 table view of the specified table is displayed. This is very useful to execute javascript code in the table view. For an example of how this can be used please see the [batch_actions\(\) hook documentation](#) .

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

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.

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 inside the folder containing your AppGini-generated application. Let's call it "example.php". Now, open that file in your text editor and paste the code below then save it.

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

    /* grant access to the groups 'Admins' and 'Data entry' */
    $mi = getMemberInfo();
    if ( in_array $mi[ 'group' ], array( '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
$currDir = dirname(__FILE__);
include("$currDir/defaultLang.php");
include("$currDir/language.php");
include("$currDir/lib.php");

/* grant access to the groups 'Admins' and 'Data entry' */
$mi = getMemberInfo();
if (in_array($mi['username'], array('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 'jogn.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
$currDir = dirname(__FILE__);
include("$currDir/defaultLang.php");
include("$currDir/language.php");
include("$currDir/lib.php");

/* grant access to all logged users */
$mi = getMemberInfo();
if ($mi['username'] != $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
    $currDir = dirname(__FILE__);
    include("$currDir/defaultLang.php");
    include("$currDir/language.php");
    include("$currDir/lib.php");

    include_once "$currDir/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 "$currDir/footer.php";
?>

```

Link to the page from other pages

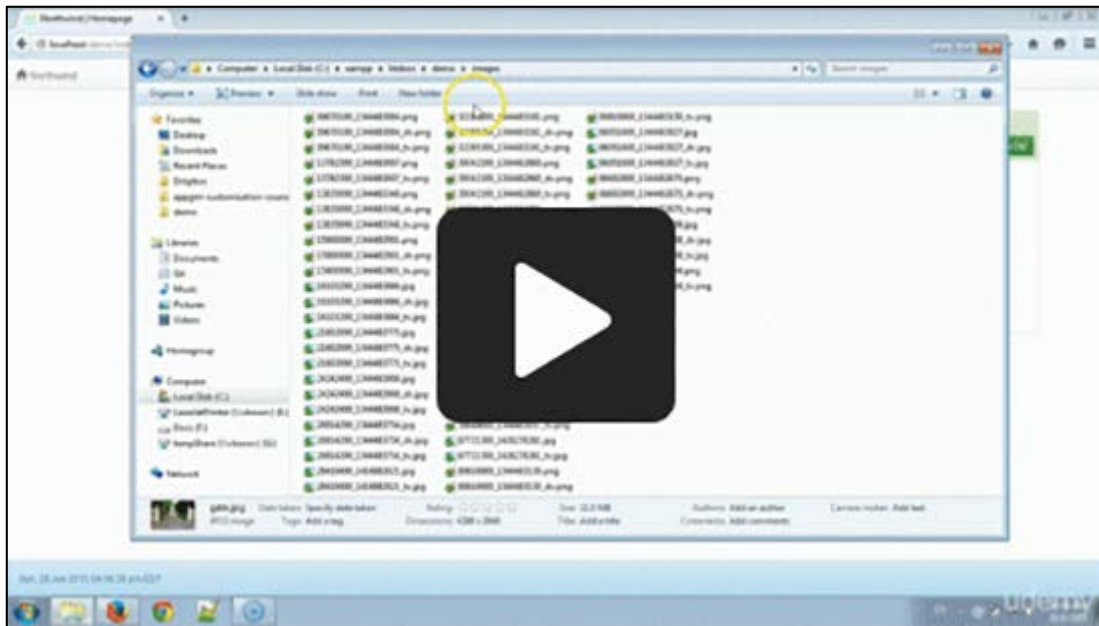
Finally, you want users to be able to easily reach your page. AppGini makes it easy to add links to the homepage and/or to the navigation menu. To do so, all you need to do is [add a few lines to the "hooks/links-home.php" and/or "hooks/links-navmenu.php" files.](#)

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.

AppGini Online Customization Course That Will Save You A Fortune!

Hooks might get tricky sometimes. That's why we prepared a course that teaches you, step-by-step, how to use hooks efficiently to expand your AppGini applications. We included 4.5 hours of video, featuring 30 common examples, with 500 lines of code, and 90 review questions.

The course is [available now on Udemy for just \\$50 \\$45](#). And you get life-time access to it.



In this course, you'll learn how to:

- customize the login page,
- add links to the main page and show the count of records,
- organize long forms into tabs or change their layout,
- change the filters page to look more friendly and efficient,
- add custom buttons to forms to extend their functionality,
- create invoices and similar report types,
- add validation rules to forms to enforce business rules,
- perform calculations on the fly,
- and automatically update an order total field as items get added to the order.



[Click here to learn more and preview some of the lessons in this course.](#)

All the code we'll write in the course will be explained in a clear easy-to-follow pace so that you can write your own code afterwards.

Moreover, we'll continue adding more and more lectures to expand the course and include additional examples.

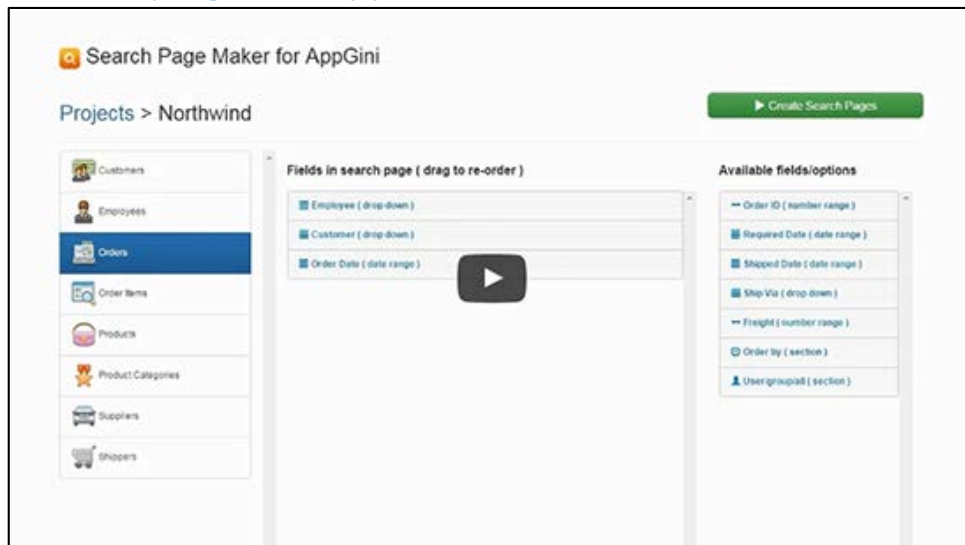
By signing up now, you'll also get all future lectures for free. You'll have unlimited access to the course as it never expires. And you can learn using your PC, tablet or mobile, anytime, anywhere.

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](#)

Search Page Maker plugin for AppGini



Search Page Maker (SPM) enables you to build user-friendly yet powerful search pages for your AppGini application by simply dragging and dropping the fields you want to include in search.



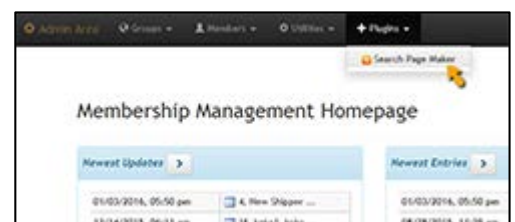
User-friendly and powerful search. SPM plugin allows you to create beautiful and user-friendly search pages, with support for date pickups, date and number ranges, auto-complete drop-downs, radio button options, and other easy-to-use search controls.

Simple search modes. Most of your application users would just love the new search pages created by SPM due to being simple yet powerful. However, the more advanced users might still want more control over search criteria. So we added a 'Switch to advanced search mode' button that switches between the SPM search page and the standard one generated by AppGini.



Fun to use! SPM plugin makes it very easy to design the search page for each of your application tables. It's as easy as dragging and dropping fields to include and order them. And you can also specify whether to show the 'Sort by' options or hide them.

Easy installation. To install SPM into any of your AppGini-generated applications, just unzip it into the application folder. Once you do this, you can access SPM from a 'Plugins' drop-down menu that will automatically show up to the admin user.



Buy SPM

Instant download link sent by email

SPM Frequently asked questions

Which version of AppGini is required for SPM?

AppGini 5.50 or above is required.

If a new version of AppGini is released, do I need to update SPM as well?

Usually not. However, if an update is needed we'll issue it free-of-charge.

If you make a new version of SPM, do I have to pay to upgrade?

No. If you buy SPM, you get free life-time updates to it.

I have several AppGini applications. Do I have to buy a separate license of SPM for each?

No. If you buy SPM, you can install it to all of your AppGini applications. However, just like AppGini, if you are a team of developers who use AppGini as part of your development toolset, each developer should have a separate license.

I noticed that SPM can automatically configure the hooks. If I have custom hooks in a project, will SPM overwrite them?

No, it will just insert a single line of code into the tablename_init hook, directly after the function definition line ... It won't affect any of your hooks code.

I have a question not addressed here ... how can I contact you?

Please use [our online support form](#).

What's new?

Latest version: 1.02, released on Apr 25, 2016

Fixed a compatibility issue with PHP 7.

Buy SPM

Instant download link sent by email

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](#).

Hosting providers

Most hosting providers support hosting the applications generated by AppGini since AppGini generates PHP applications that connect to MySQL databases. PHP and MySQL are very widely available through almost any hosting provider. Personally, we've been using [innohosting](#) for several years and are quite content with their support and reliability. A good source of information regarding hosting providers is the [webhostingtalk forum](#).

Productivity tools

These may not be related directly to AppGini, except for the fact that we found them to be greatly effective time savers.

- [KeyBreeze](#): An open source utility that allows you to access any folder, application or URL by typing part of its name (or any other name you give to it) without opening any browser windows or using the mouse to navigate to any menus. Once you use it, you'll become addicted to it!
- [Boomerang](#) is a browser add-on that plugs into your Gmail inbox and allows you to schedule emails for sending later and reminds you of emails at a later date/time. It's a great tool for organizing your email and removing the noise. There is a free limited version which is enough for most uses, plus a more advanced paid version.

Code management

If you plan to manually edit the generated code, you should be careful to avoid overwriting your modifications if 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 [TortoiseSVN](#).

Reference material

- [HTML reference](#)
- [CSS reference](#)
- [PHP reference](#)
- [MySQL SQL reference](#)