
Gila CMS Documentation

Release 1.7.2

Vasileios Zoumpourlis

Oct 27, 2018

| | | |
|----------|------------------------------------|-----------|
| 1 | About | 3 |
| 1.1 | Why choose Gila? | 3 |
| 1.2 | Spreading the word! | 3 |
| 1.3 | Where you can get help | 3 |
| 2 | Installation | 5 |
| 2.1 | Preparation | 5 |
| 2.2 | Installer | 5 |
| 3 | Content | 9 |
| 3.1 | Pages | 9 |
| 3.2 | Posts | 11 |
| 3.3 | Categories | 13 |
| 3.4 | Media | 13 |
| 4 | Administration | 15 |
| 4.1 | Users | 15 |
| 4.2 | Main Menu | 16 |
| 4.3 | Widgets | 16 |
| 4.4 | Packages | 17 |
| 4.5 | Themes | 18 |
| 4.6 | Settings | 19 |
| 5 | Structure | 23 |
| 6 | Packages | 25 |
| 7 | Examples | 27 |
| 7.1 | Event: Post Tags | 27 |
| 7.2 | Widget: Twitter Timeline | 28 |
| 7.3 | User: Address | 29 |
| 8 | Themes | 31 |
| 8.1 | New theme | 31 |
| 8.2 | Child themes | 32 |
| 8.3 | Cloned themes | 32 |
| 9 | Schemas | 33 |

| | | |
|-----------|---------------------------|-----------|
| 9.1 | package.json | 33 |
| 9.2 | Table Schema | 34 |
| 9.3 | Field Schema | 35 |
| 10 | Api | 37 |
| 10.1 | Class gila | 37 |
| 10.2 | Class event | 41 |
| 10.3 | Class view | 41 |
| 10.4 | Class db | 44 |
| 11 | Indices and tables | 47 |

Contents:

Gila CMS is an open-source and free content management system built with php7. Built with MVC architecture, is very easy to develop on it any customized solution. It is licensed under BSD 3-Clause License. The website is gilacms.com

1.1 Why choose Gila?

Gila CMS is a good option for self-hosted blogs or startup websites

- Installs by default a blogging system with social integrations.
- No coding skills are required to install or maintain the website
- Themes that include are responsive, that makes it accessible in all devices.
- It is fast, and compresses the content where it needs to.

1.2 Spreading the word!

You can help us spread the word about Gila CMS! We would surely appreciate it!

- Follow our [Facebook Page](#)
- [Retweet us!](#)
- Give a star on [Github](#)

1.3 Where you can get help

- Join [Slack](#) or [Gitter](#)
- Ask on [stackoverflow](#) using the tag **gilacms**

2.1 Preparation

Before beginning with installation make sure that your web host or local server meets these requirements:

- Apache 2 server
- MySQL / MariaDB server
- PHP 5.4+ with the following extensions *mysqli*, *zip*, *mysqlnd*, *json*, *gd* and *mod_rewrite* enabled

If you want to install gila cms in your local machine and not sure how to prepare your server don't hesitate to ask for help on [Slack](#)

First unzip gila in a public html folder e.g */var/www/html/gila* and make sure that the folder is writable from the application.

On **nginx** server you will need to configure the redirects in */etc/nginx/sites-enabled/default* ([issue #1](#))

```
location / {
    index index.php index.html index.htm;
    rewrite gila/(?!install) (?!src) (?!themes) (?!lib) (?!assets) (?!tmp) (?!robots.txt) (.
↪*)$ /gila/index.php?url=$1 last;
}
```

In order to proceed with the installation, you will need your **database settings**. If you do not know your database settings, please contact your host and ask for them. You will not be able to continue without them. More precisely you need the database hostname, the database name, the database username and password.

2.2 Installer

We access in installation page with the browser e.g *http://localhost/gila/install*

Gila CMS Installation

| | |
|--|--|
| Hostname | Admin Username |
| <input type="text" value="localhost"/> | <input type="text"/> |
| Database | Admin Email |
| <input type="text"/> | <input type="text"/> |
| DB Username | Admin Password |
| <input type="text"/> | <input type="text"/> |
| DB Password | Base URL |
| <input type="text"/> | <input type="text" value="//localhost/gila/"/> |
| <input type="submit" value="Submit"/> | |

In the installation page we must fill all the fields

- **Hostname:** the hostname of the database, usually it is *localhost*
- **Database:** name of the database
- **DB Username, DB Password:** the username and the password in order to connect to the mysql
- **Admin Username, Admin Email, Admin Password:** a user will be created for the website as administrator with these data
- **Base Url:** the web address of the website must finish with *'/'* e.g. *mywebsite.com/*

After filling the data and submit them, we wait a few seconds until the installation is finished.

Installation finished successfully!

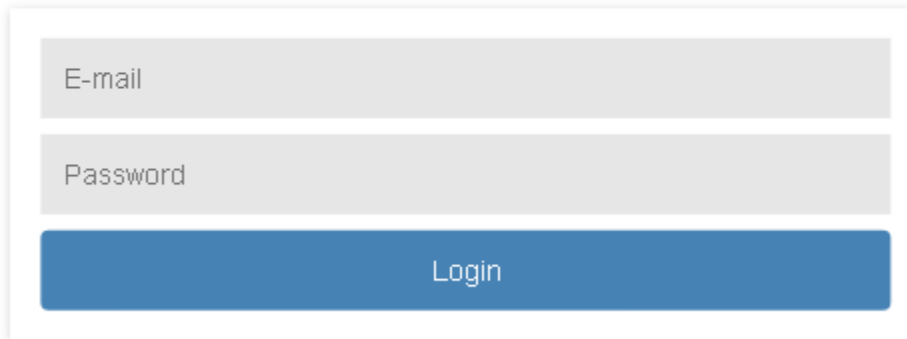
Visit the website

Login to admin panel

When installation is finished we can enter on the admin panel using the admin email and password that we wrote before.



Login



E-mail

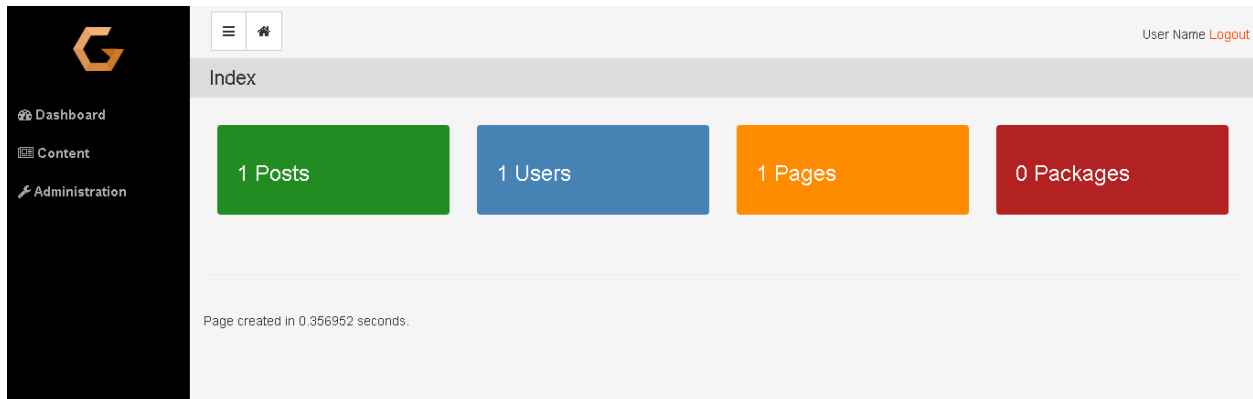
Password

Login

[Forgot password?](#)

We can always access in the login page from these links *mysite.com/ /login* it redirects to the front page of the website *mysite.com/ /admin* it redirects to the administration

We enter in the administration dashboard.



The screenshot shows the administration dashboard. On the left is a dark sidebar with the Gila CMS logo and three menu items: Dashboard, Content, and Administration. The main content area has a top bar with a hamburger menu icon, a user profile icon, and the text 'User Name Logout'. Below this is a section titled 'Index' containing four colored boxes: a green box for '1 Posts', a blue box for '1 Users', an orange box for '1 Pages', and a red box for '0 Packages'. At the bottom of the main area, it says 'Page created in 0.356952 seconds.'

On the administration menu we see two submenus

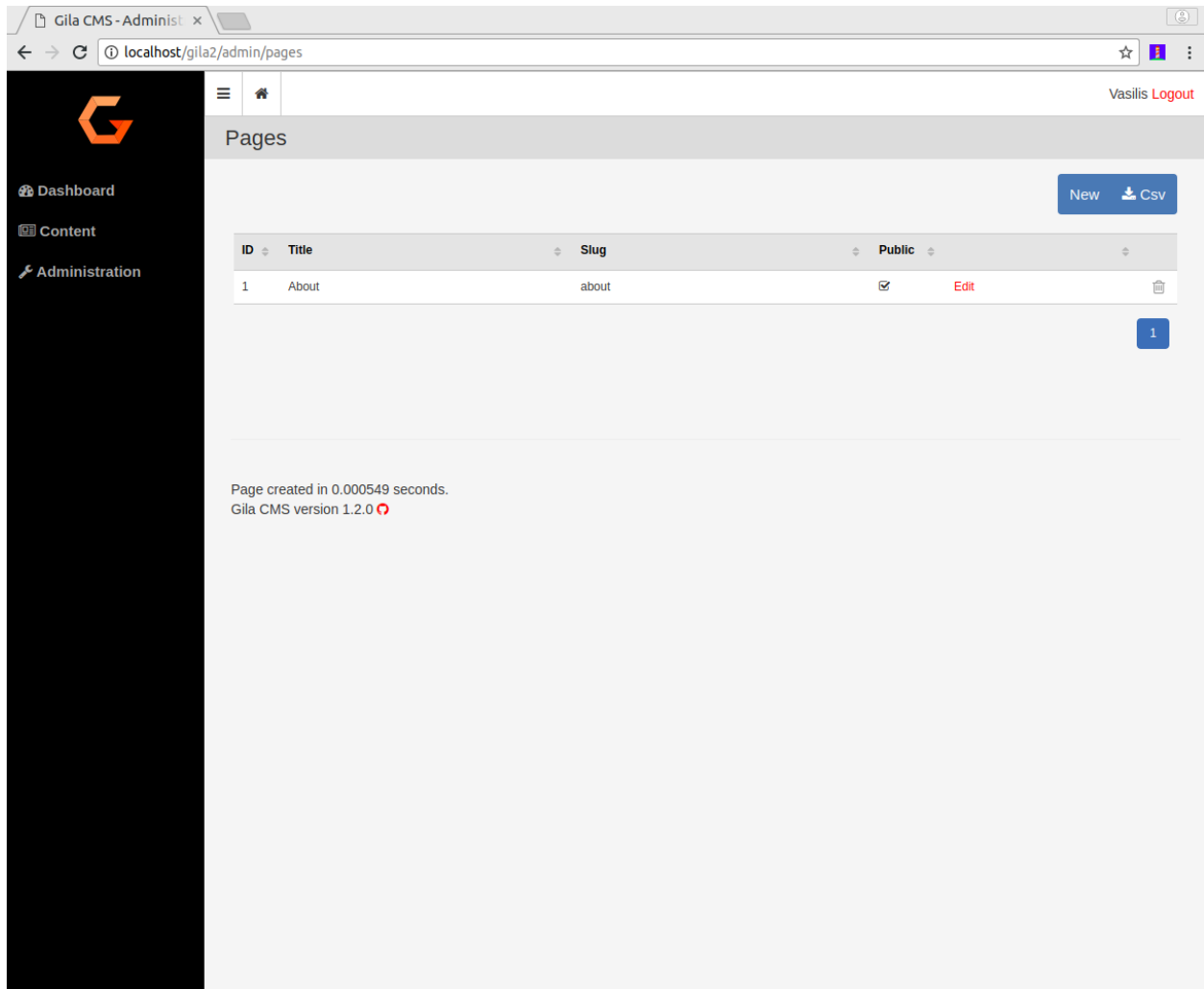
- **Content** to add or edit content like pages, posts or files
- **Administration** to edit users or change the settings of the website.

In the administration menu the Content option gives a submenu of the basic content types of Gila:

- *Pages*
- *Posts*
- *Categories*
- *Media*

3.1 Pages

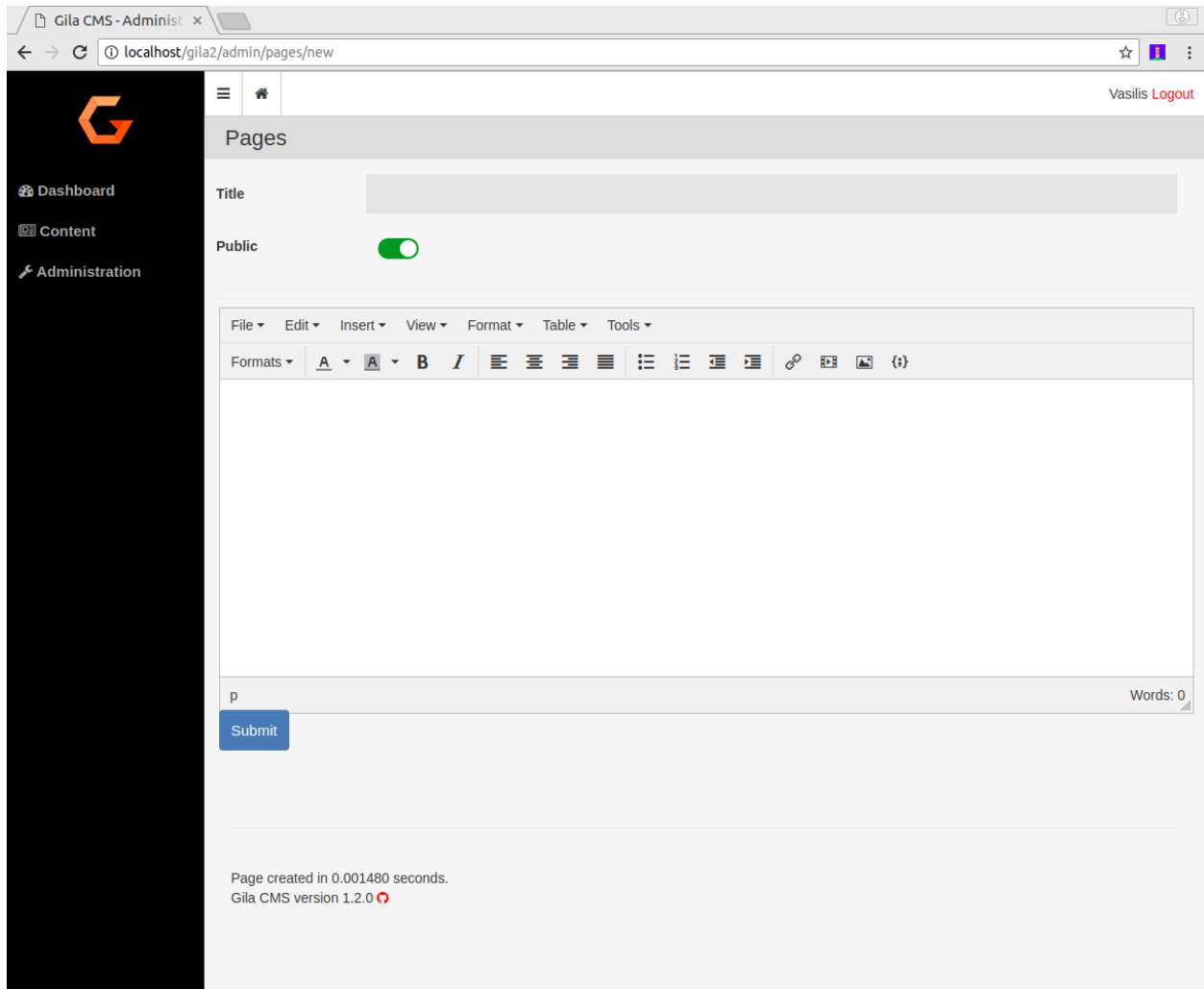
Pages are the basic content type. A page can be just a text or have media. The information of a page is independent of time so you want them to be found by the visitor in the same place, like on the menu of the website.



Every page have four values:

- **ID:** a unique identifier
- **Title:** the title of the page
- **Slug:** is the path of the page. For example the path of a page with title 'My Page' will be *mysite.com/my-page*
- **Public:** an on/off flag. If Public value is off for 'My Page' then *mysite.com/my-page* wont be accessible from the browser.

To **create a new page** click on button **New** that you see on the up-right corner of the table.



The screenshot shows the Gila CMS Admin interface for creating a new page. The browser address bar shows `localhost/gila2/admin/pages/new`. The user is logged in as Vasilis. The interface includes a sidebar with navigation options: Dashboard, Content, and Administration. The main content area is titled "Pages" and contains a form with the following elements:

- Title:** A text input field.
- Public:** A toggle switch that is currently turned on (green).
- Rich Text Editor:** A WYSIWYG editor with a menu bar (File, Edit, Insert, View, Format, Table, Tools) and a toolbar with various formatting options (bold, italic, underline, list, link, image, etc.). The editor contains the text "p".
- Words:** A counter showing "Words: 0".
- Submit:** A blue button to save the page.

At the bottom of the page, it displays: "Page created in 0.001480 seconds. Gila CMS version 1.2.0".

3.2 Posts

The posts can be news or articles about your business or the interests of the website. They are organized in categories and are listed in chronological order.

The screenshot shows the Gila CMS Admin interface. On the left is a dark sidebar with the Gila logo and navigation links for Dashboard, Content, and Administration. The main content area is titled 'Posts' and features a search bar with 'Title' and 'User' filters, a search button, and 'New' and 'Csv' buttons. Below the search bar is a table with one row of data. The table has columns for ID, Title, User, Last updated, Public, and actions (Edit and delete). At the bottom of the page, it says 'Page created in 0.000939 seconds. Gila CMS version 1.2.0'.

| ID | Title | User | Last updated | Public | |
|----|-------------|---------|---------------------|-------------------------------------|----------------------|
| 1 | Hello World | Vasilis | 2017-11-19 19:21:44 | <input checked="" type="checkbox"/> | Edit |

To create a new post click on button **New** that you see on the up-right corner of the table.

| | |
|--------------------|--|
| Title | <input type="text"/> |
| Slug | <input type="text" value="Generate from title"/> |
| Description | <input type="text" value="A small description of 200 characters"/> |
| Thumbnail | <input type="text" value="Thumbnail image"/> |
| Categories | <input type="text"/> |
| Tags | <input type="text" value="values seperated by comma"/> |
| Public | <input checked="" type="checkbox"/> |

File Edit View Insert Format Table

Formats A A B I [List Icons]

0 WORDS POWERED BY TINYMCE

Submit

3.3 Categories

Categories are used to categorize posts or maybe other popular content that you could use later. You only add or edit the names of the categories.

3.4 Media

Media are the images that you want to use for your posts. They are saved as files and not in the database like the other content types. The root directory of media is */assets*. The files and subfolders of */assets* are visible in the public by the path *mysite.com/assets* so you should not upload files or images that you don't want to be found from search engines.

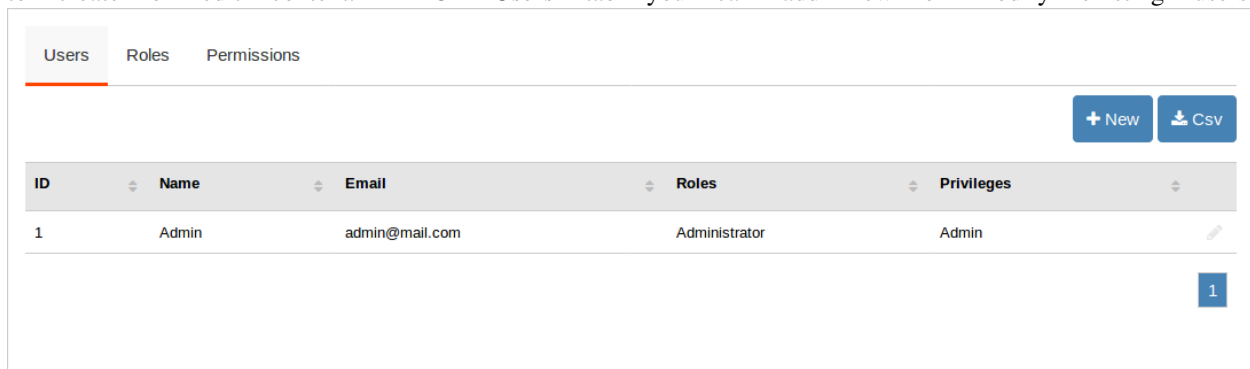
The screenshot shows the Gila CMS Admin interface. The browser address bar displays 'localhost/gila/admin/media'. The user is identified as 'Vasilis Zoumpourlis'. The main content area is titled 'Media' and shows a gallery of 18 image files. Each file is represented by a thumbnail and a filename starting with 'assets/media/'. The files are arranged in two rows of nine. The first row contains files /1.jpg through /17.jpg. The second row contains files /18.jpg through /8.jpg, followed by a file with a long alphanumeric name: /872636686782ffe27h4h. A 'Choose Files' button and the text 'No file chosen' are visible at the top of the gallery. On the left side, there is a dark sidebar with navigation links: Dashboard, Content, Administration, Tasks, Exoplanets, and File Manager. At the bottom of the page, there is a footer with the text: 'Page created in 0.001472 seconds. Gila CMS version 1.3.0'.

In the administration menu you the Administration option gives a sub menu of the basic administration areas

- *Users*
- *Main Menu*
- *Widgets*
- *Packages*
- *Themes*
- *Settings*

4.1 Users

Users are the persons that you can grant access to website and give them privileges to create or edit content. On *Users* tab you can add new or modify existing users.



The screenshot shows a web interface for managing users. At the top, there are three tabs: 'Users' (selected), 'Roles', and 'Permissions'. Below the tabs, there are two buttons: '+ New' and 'Csv'. Below the buttons is a table with the following columns: ID, Name, Email, Roles, and Privileges. The table contains one row with the following data: ID: 1, Name: Admin, Email: admin@mail.com, Roles: Administrator, Privileges: Admin. There is a small blue box with the number '1' in the bottom right corner of the table area.

| ID | Name | Email | Roles | Privileges |
|----|-------|----------------|---------------|------------|
| 1 | Admin | admin@mail.com | Administrator | Admin |

4.1.1 User Roles

On *Roles* you can add new roles for the users. Roles represent job functions and through them users acquire specific

| ID | Role |
|----|---------------|
| 3 | Developer |
| 2 | Publisher |
| 1 | Administrator |

permissions.

4.1.2 Permissions

On *Permissions* you can set the common permissions that all registered users can have or to specific user roles. Permissions grant to the users access to resources or perform certain operations.

| | Member | Administrator | Publisher | Developer |
|--|--------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| admin All administration privileges | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| admin_user Administrate users | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| admin_userrole Administrate user roles | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> |
| admin_permissions Add and remove permissions to user roles | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> |
| upload_assets Can upload files on assets folder | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| edit_assets Can edit files on assets folder | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> |

4.2 Main Menu

The main menu is displayed in the navigation bar of the front pages of the website. By default the menu will include a link to the home page, the post categories and the publish pages. In this area you can modify the with the menu editor. This [video](#) demonstrates quickly how you can edit the menu:

4.3 Widgets

Widgets are some blocks that you can show them on the layout of the website and improve the user experience of the visitors. Widgets can be for example *menus*, *comment sections*, *text blocks*, *lists of links*. Every page have four values:

- **ID**: a unique identifier

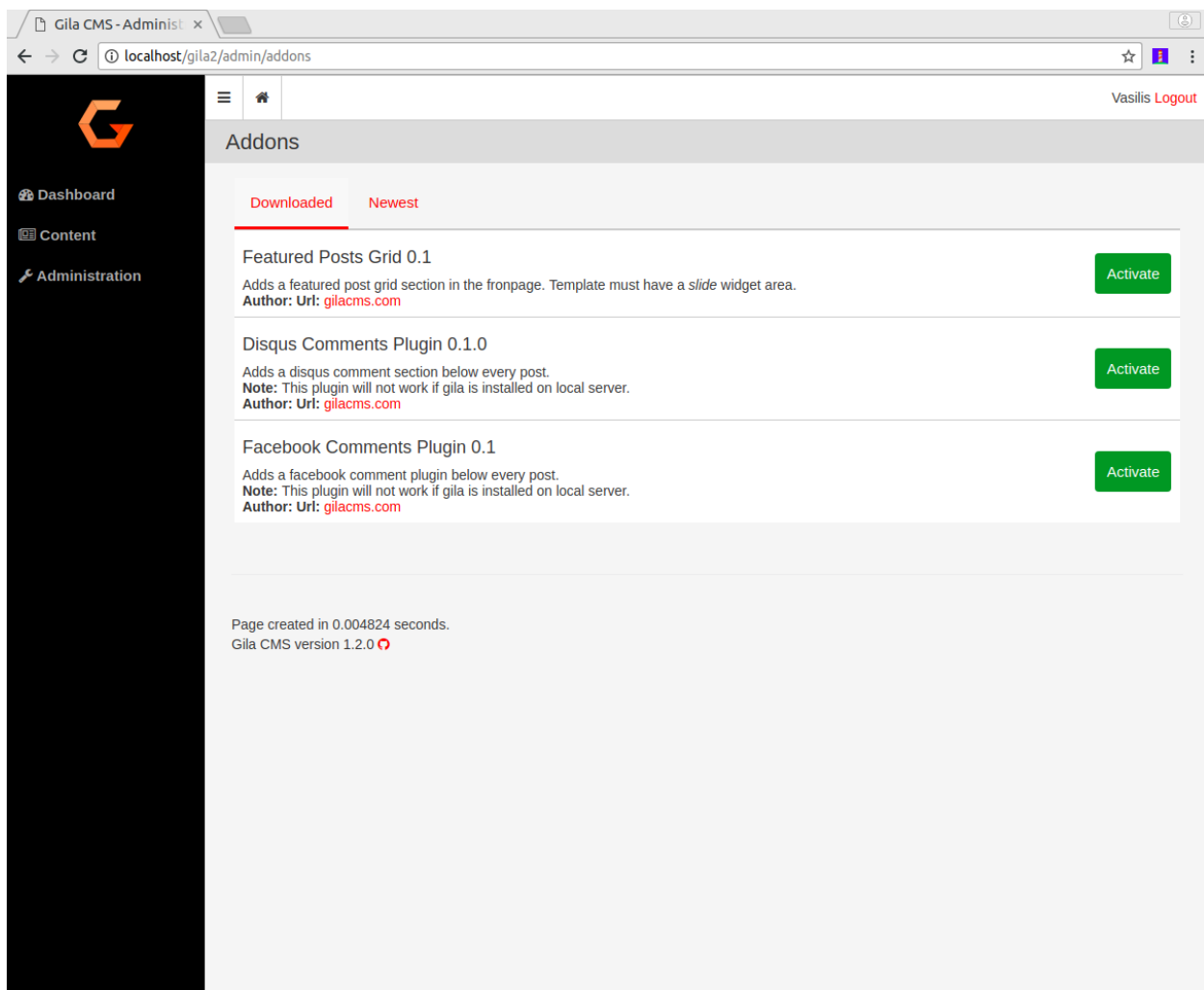
- **Title:** the title of the widget
- **Type:** widget type
- **Widget Area:** is where the widget will be displayed. Can be an area in a view from the website theme or the administration.
- **Position:** the position of the widget in the widget area.
- **Active:** if the widget is visible or not.

Notice: When you create a new widget you can set the type of the widget, this field cannot not be changed later since it determines the widget's structure of the data.

4.4 Packages

Packages give new functionalities on your web application. They may add a specific widget, a few new links in the administration menu or add new content and new templates to show the content. For example *Facebook Comments Plugin* add a facebook comments section below every page post. *Featured Posts Grid* show the thumbnails photos from featured posts in the front page of a blog theme.

You can administrate packages from Administration->Packages

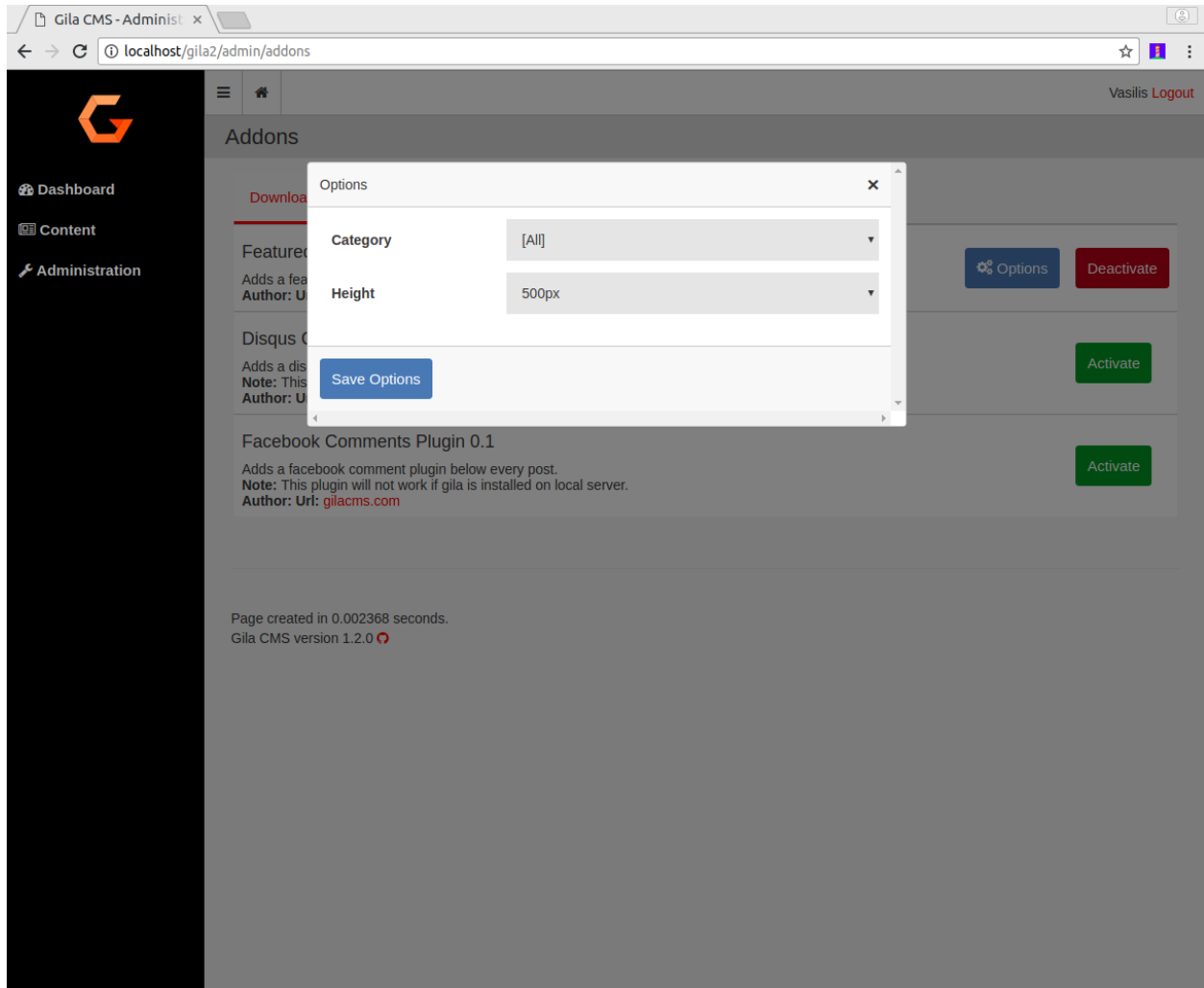


The screenshot shows the Gila CMS Administration interface. The browser address bar indicates the URL is localhost/gila2/admin/addons. The user is logged in as Vasilis. The main content area is titled 'Addons' and has two tabs: 'Downloaded' (selected) and 'Newest'. There are three addons listed:

- Featured Posts Grid 0.1**: Adds a featured post grid section in the fronpage. Template must have a *slide* widget area. Author: gilacms.com. [Activate](#)
- Disqus Comments Plugin 0.1.0**: Adds a disqus comment section below every post. **Note:** This plugin will not work if gila is installed on local server. Author: gilacms.com. [Activate](#)
- Facebook Comments Plugin 0.1**: Adds a facebook comment plugin below every post. **Note:** This plugin will not work if gila is installed on local server. Author: gilacms.com. [Activate](#)

At the bottom of the page, it says: Page created in 0.004824 seconds. Gila CMS version 1.2.0

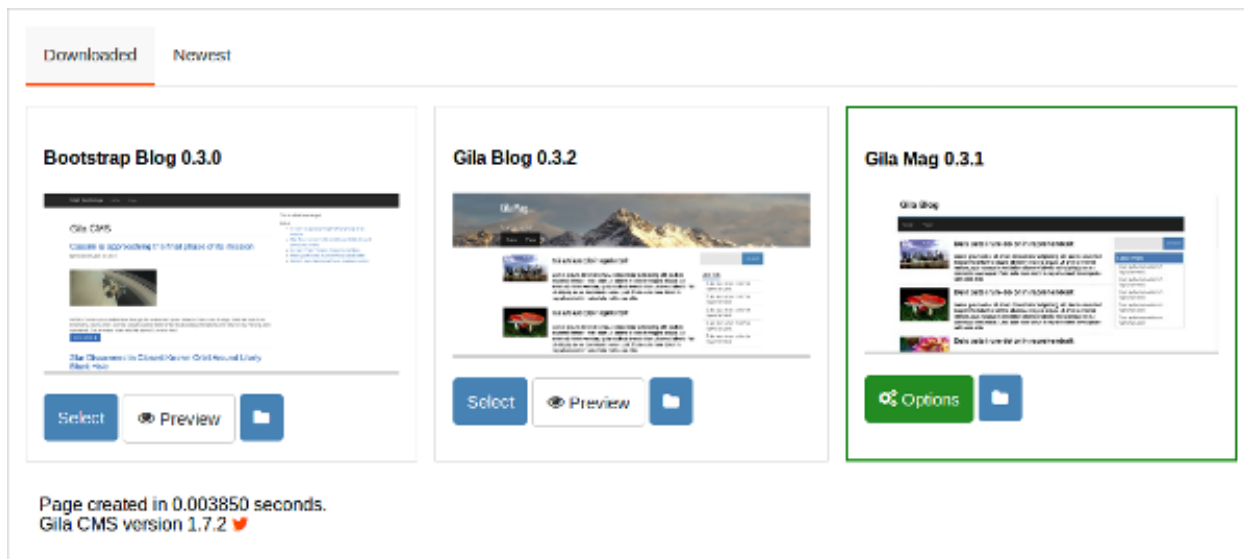
The installed packages usually show an **Options** button. By clicking this button you can change some parameters for the specific package. When you save the settings the changes will take effect by reloading the page.



4.5 Themes

Themes change the look and style of your website. They use different colors and fonts and helps your visitors identify your website and improve their user experience (UX).

You can select the theme from Administration->Themes



The selected theme usually shows an **Options** button. By clicking this button you can change some options for the theme like the header image (logo) of the website or the main color.

4.6 Settings

On Administration->Settings page and we can make the following configurations

4.6.1 Basic Settings

Basic Settings

| | |
|-------------------------------|---|
| Title | <input type="text" value="Gila CMS"/> |
| Description | <input type="text" value="An awesome website!"/> |
| Website URL | <input type="text" value="http://localhost/gila/"/> |
| Admin Email | <input type="text"/> |
| New Users Can Register | <input type="checkbox"/> |
| Theme | <input type="text" value="Limon"/> |
| Timezone | <input type="text" value="America/Mexico_City"/> |
| Language | <input type="text" value="English"/> |
| Admin Logo | <input type="text" value="assets/index.jpeg"/> |

Advanced Settings

| | |
|---------------------------|--|
| Use CDN | <input type="checkbox"/> |
| Pretty Urls | <input checked="" type="checkbox"/> |
| Default Controller | <input type="text" value="Blog"/> |
| Environment | <input type="text" value="Development"/> |
| Check For Updates | <input type="checkbox"/> |

- **Title** is the website title. It will appear up from the menu if we don't use a logo from the theme options.
- **Description** is a small text that describes the website.
- **Website URL** the url path for exmple 'https://mysite.com'
- **Admin Email** sets the email of the administration.
- **New users can register** adds the registration form for the visitors so they can register as users.
- **Theme** changes the look and style of your website. You can also change the theme from *Administration->Themes*
- **Timezone** The dates and times saved in posts, logs and the rest of the content will be based on the selected

timezone.

- **Language** The language of the website and the administration menu.
- **Admin Logo** set the image to display to administration menu and login screen.

4.6.2 Advanced Setting

You change these setting if you are developing or set up the installtion.

- **Use CDN** Use CDN for static files of popular libraries (jquery.js, vue.js). It's not advised for local installation where internet connection may fail.
- **Pretty Urls** turns `?c=blog&action=tag&tag=linux` into `blog/tag/linux`. If is not selected by default then your apache server may not have the `mod_rewrite` enabled.
- **Default controller** The controller that will be used if the calling path do not provide it as first parameter. For example the **Admin** controller is used when we call `mysite.com/admin` but when we call `mysite.com` or `mysite.com/my-post` the default controller will be used, which is **Blog**, so these paths are equal with `mysite.com` and `mysite.com/my-post`. There is not need to change the default controller unless you want to change how the website will be used.
- **Environment** If changes to *Development* the website wont use the combine `load.php` from the packages and will display all notices and errors of the back end. Must use it when you make changes in the code.
- **Check For Updates** will automatically search for new updates on packages and display alerts.

In the main folder we can see these folders and files.

```
assets/  
lib/  
log/  
tmp/  
src/  
themes/  
index.php  
config.php
```

assets/ A public folder where we upload our media files.

lib/ A public folder. Third-party libraries are inside this folder.

log/ A private folder that save logs and the user sessions.

tmp/ A public folder with temporally files created.

src/ The folder of installed packages. Here is all the code of the system.

themes/ The folder of installed themes.

index.php The main index file. For any call, execution starts from here.

config.php The configuration file. It is generated after installation.

Packages

The source code of Gila CMS is split into packages, even the core files are part of the main package called *core*. The package folders are placed inside *src/* folder and desirably have a similar structure:

```
assets/  
controllers/  
models/  
views/  
lang/  
package.json  
load.php
```

The folders are optional but very useful to organize better the code. The file **package.json** is a must have as it has the basic information of the package -without it the package is invisible- and the **load.php** is the file that will register new values and events of the package.

A simple **package.json** file:

```
{  
  "name": "Package Name",  
  "version": "1.0.0",  
  "description": "A short descriptive text of your package for what it does.",  
  "url": "package_url.com",  
  "author": "Your Name"  
}
```

You can also add another index in the object called *options*. It will be an array of objects, the objects are the options to be stored. The index is the option name and it can have optional values with the following indexes:

- **title** the option name to be displayed, if not specified, the index is used
- **type** select | postcategory
- **options** array of {value:display_text}, it is required if is set type:select

```
{
  ...
  "options":{
    "option1":{},
    "category":{
      "type":"postcategory"
    },
    "lang":{
      "title":"Language",
      "type":"select",
      "options":{
        "en":"English","es":"Spanish","el":"Greek"
      }
    }
  }
}
```

You can get the option values like that:

```
// options are saved using as prefix the package's folder name
// for example if the package has the folder my_package/

$option1 = gila::option("my_package.option1");
$lang = gila::option("my_package.lang","en"); // use default value 'en'
```

More information for [package.json schema](#).

A simple **load.php** file could be:

```
<?php
// display text below any post
event::listen('post.after',function(){
    echo 'This is printed after post.';
})
```

IMPORTANT: The first line of the load.php file should include only the opening tag `<?php` and not close with the closing tag.

Here are some examples to you get into the of package creation.

7.1 Event: Post Tags

In this example we will list the tags of a post just after it. Create a folder inside *src/* and name it *post-tags*. Inside it create the following files

```
package.json  
load.php  
logo.png
```

package.json

```
{  
  "name": "Post Tags",  
  "version": "1.0.0",  
  "description": "A package to show tags below the post."  
}
```

load.php will run a code when the package is active. We will register a function to run right after the post display.

```
<?php  
event::listen('post.after', function(){  
    global $g; // $g will give us the post id  
    $tags = core\models\post::meta($g->id, 'tag'); // get the tag list of post  
    echo "<strong>TAGS:</strong> ";  
    foreach ($tags as $tag) echo " <a href='tag/$tag'>#$tag</a>";  
});
```

This function will run when the *post.after* event is dispatched. That happens with *event::fire('post.after')*; or *event::widget_area('post.after')*; from *single-post.php* view file.

logo.png is the package's logo and is displayed in the package list.

Activate the package in */admin/packages*. After that you should see the list of TAGS below any blog post.

7.2 Widget: Twitter Timeline

In this example we will create a widget that displays the last tweets of an account. Instead of using an event to run the code we let the user create instances of the widget choose in which widget area want to display the twitter plugin. Inside *src/* create a folder *twitter-timelines* and add the following files:

```
package.json
load.php
widgets/twitter-timeline/widget.php
widgets/twitter-timeline/twitter-timeline.php
```

package.json:

```
{
  "name": "Twitter Timelines",
  "version": "1.0.0",
  "description": "Installs a widget to display twitter timelines."
}
```

load.php:

```
<?php

// registers the widget name and its path
gila::widgets([
  'twitter-timeline' => 'twitter-timelines/widgets/twitter-timeline'
]);
```

widgets/twitter-timeline/widget.php will include the widget options we want to use. In this case we need the user account and the name to be displayed.

```
<?php

$options=[
  'accountID'=>[
    'title'=>'Twitter Account'
  ]
];
```

widgets/twitter-timeline/twitter-timeline.php is the view file of the widget, it will generate the html code. We use the embedding Twitter content from [here](#)

```
<?php
$account = gila::option('twitter-timelines.accountID', 'gilacms');
?>
<a class="twitter-timeline" data-height="400" href="https://twitter.com/<?=$account?>
↪">Tweets by <?=$account?></a>
<script async src="https://platform.twitter.com/widgets.js" charset="utf-8"></script>
```

gila::option() gets the option of the package that we set up in the package settings. A default value can be used if the option is null.

Activate the package. Now in `/admin/widgets` you can create a new widget with type `twitter-timeline` and set the widget area `sidebar` or `dashboard` to see it.

7.3 User: Address

In this example we will add a new field for the users. Instead of adding a new column in the database table we will use the `usermeta` table to store new values that link to the users. Inside `src/` create a folder `user-address` and add the following files:

```
package.json
load.php
```

package.json:

```
{
  "name": "User Address",
  "version": "1.0.0",
  "description": "Adds a new field for the users."
}
```

load.php:

```
<?php

// make changes to the user content type

gila::contentInit('user', function(&$table){
    $table['fields']['useraddress'] = [
        'title'=>"Address", //the label
        'type'=>'meta', //the values of the field will be stored in a meta table
        'input-type'=>'text', //use the text input type
        'edit'=>true, //is editable from user
        'values'=>1, //this field gets only one value
        "mt"=>['usermeta', 'user_id', 'value'], //meta table, meta column that links
↳to user table,meta column of the value
        'metatype'=>['vartype', 'address'] //meta column of the value type, value type
    ];
});
```

This package will add a new new Address field for users in *Administration->Users*

Themes change the look and style of your website. You can also change the theme from *Administration->Themes*. They are located in **themes/**.

What themes do

- They override the view files from the packages. View files in packages are placed in a **views/** folder, in themes are placed directly in their root folder. For example when a controller will try to render a view file from the package core it will call **view::renderFile('blog-post.php','core')**. If exists the file **themes/<selected-theme>/blog-post.php** it will rendered instead of **src/core/views/blog-post**
- They override the response files of the widgets. These files are placed in **widgets/** folder of the theme. For example, **themes/<selected-theme>/widgets/text.php** will replace **src/core/widgets/text/text.php** when a widget of text will be rendered. This is useful when you want to add styling to widgets similar to the theme. Note that you don't have to create folders for each widget.

Structure Inside a theme folder we need to have the following files:

```
package.json  
screenshot.png  
load.php  
widgets/
```

- **package.json** has the basic information of the theme and uses a similar schema with the packages.
- **screenshot.png** can be an image of maximum width of 320 pixels. It is used to display the theme on the the themes selection.
- **load.php** registers new variables to the system, like the widget areas that offers, routes for static files or a parent theme.
- **widgets/** folder include the responses of the widgets that you want to override.

8.1 New theme

How to create new theme:

Create a new folder inside **themes/** with name *my-new-theme*. Inside create a new *package.json* file:

```
{
  "name": "My New Theme",
  "version": "1.0.0",
  "author": "My Name"
}
```

Now you have a new but empty theme to choose in *Administration->Themes*. It is only missing is a *screenshot.png* file to display.

Although it has no view files of its own, when is selected, it will use the original view files of the **src/core/views/** folder.

8.2 Child themes

Child themes is the easiest way to make a new theme by making just a few changes on an existing theme. Make changes directly on another theme is a bad idea, since it can be updated any time and you lose all the adjustments you made.

How to create a child theme:

We choose *gila-blog* as parent, first create a new folder inside **themes/** with name *my-child-theme* and a *package.json*:

```
{
  "name": "My Child Theme",
  "version": "1.0.0",
  "author": "My Name",
  "parent": "gila-blog"
}
```

We create *load.php* file:

```
<?php
view::$parent_theme = 'gila-blog';
... copy here the code of themes/gila-blog/load.php
```

We need to let view class know which is the parent theme, so first the rendering methods will try to find the view files in the parent theme when they fail to find theme in the selected(child) theme.

Now inside the folder of this theme you can edit the view files. Giving to the folder a name with prefix **my-** or **custom-** you make sure that the name will not conflict with a public theme from theme manager.

8.3 Cloned themes

If you want to create a new theme based on another one you can just copy its files. In this way you dont have to keep installed the original theme.

How to clone a theme:

Make a copy of *gila-blog* and name it *my-clone-theme*. In *package.json* file change the name:

```
...
  "name": "My Clone Theme",
  ...
```

9.1 package.json

package.json is used from both packages and themes to set their values. This schema is daved in json format and uses these indeces:

- **name** (string)

Name of the package or theme.

- **version** (string)

Version in (semantic versioning)[<https://semver.org/>]

- **description** (string)

A small paragraph explaining what the package does.

- **url** (string)

A path to more information for this package.

- **logo** (string)

An image path to display as logo. Used only by packages.

- **screenshot** (string)

An image path to display as screenshot. Used only by themes.

- **parent** (string)

The parennt package. Used only by themes.

- **options** (object)

An array of options that uses this package. They use the *Field Schema*.

- **permissions** (object)

An image path to display as logo. Used only by packages.

Example

```
"permissions": {
  "admin": "All administration privileges",
  "admin_user": "Administrate users"
}
```

- **lang** (string)

A relative path to the language prefix that translates the strings set in the package.json like permissions and option labels.

9.2 Table Schema

The table schema is used for a content type. It gives to the application the structure of the database table that stores the data. So the content administration page will be generated on its own and the content creators can manipulate the data with no further code.

- **name** (string)

The name of database table that data is stored.

- **title** (string)

The title to display at content administration.

- **id** (string)

The field index that is the primary key on database table. Default value is *id*.

- **permissions** (assoc array)

Associative array of user permissions required to run the actions. Example:

```
'permissions'=>[
  'create'=>['admin', 'content-contributor'],
  'update'=>['admin', 'content-editor'],
  'delete'=>['admin']
],
```

- **fields** (assoc array)

Associative array of the content fields. They follow the *Field Schema*. Example:

```
'title'=> [
  'title'=>'Title',
],
```

- **pagination** (int)

Number of results per page in content administration.

- **lang** (string)

A relative path to the language prefix that translates the strings set in the table schema like permissions and field titles.

- **search-box** (boolean)

If true, it displays a search box in content administration.

- **tools** (array)

An array of tools that will be displayed in content administration.

- **commands** (array)

An array of commands that will be displayed in content administration.

- **search-boxes**

An array of field names. Their search filters will be displayed in content administration as.

- **children** (assoc array)

References to other content types that are partials of the parent content. The index of a child must be an existing content type. The child is an associative array with two indeces: - **parent_id** (string) The field of child table that points to the parent's id. - **list** (array) The listed fields of child table. The schema of the child must result in the same list of fields.

Example child for a *shop_order* content type:

```
'children'=>[
  'shop_orderitem'=>[
    'parent_id'=>'order_id',
    'list'=>['id', 'image', 'product_id', 'description', 'qty', 'cost']
  ]
]
```

- **events** (array of [string,function])

The first value is the event name and the second value is the function that will be triggered. The function gets a reference to the specific row of the table. Possible values for the event: - **change** Runs when a row is created or updated

9.3 Field Schema

Fields are used as options from packages and widgets or as columns from table schemas. When are used as options from packages their format is JSON, the other cases are as php associative arrays.

[Field Schema is still unfinished, please join a chat for more information]

- **title** (string)

The label of the field to display.

- **default** (any)

The default value to use in input field.

- **type** (string)

The field type. Specifies how the data is processed. If *input-type* is not specified it will also be used as input type. These values can be: - text - select - meta

- **input-type** (string)

Specifies what input type will be used. Default values for 1.8.0: - select - meta - radio - postcategory - media - textarea - tinymce - checkbox - switcher - list (cannot be used in table schemas)

- **allow-tags** (boolean/string)

Lets the field value keep html tags or remove them. The default value of *allow-tags* is false.

Example

```
"allow-tags": "<a><p><ul><li>",
```


10.1 Class gila

class gila

Common methods for Gila CMS

controller (*\$c*, *\$file*, *\$name=null*)
(static) Register a new controller.

Parameters

- **\$c** (*string*) – Controllers name
- **\$file** (*string*) – Controller's filepath without the php extension
- **\$name** (*string*) – Optional. Controller's class name, \$c is used by default

Example:

```
gila::controller('my-ctrl', 'my_package/controllers/ctrl', 'myctrl');
```

route (*\$r*, *\$fn*)
(static) Registers a function call on a specific path.

Parameters

- **\$r** (*string*) – The path
- **\$fn** (*Function*) – Callback for the route

Example:

```
gila::route('some.txt', function(){ echo 'Some text.'; });
```

onController (*\$c*, *\$fn*)
(static) Registers a function to run right after the controller class construction.

Parameters

- **\$c** (*string*) – The controller’s class name
- **\$fn** (*Function*) – Callback

Example:

```
gila::route('blog', function(){ blog::ppp = 24; });
```

action (*\$c, \$action, \$fn*)

(static) Registers a new action or replaces an existing for a controller.

Parameters

- **\$c** (*string*) – The controller’s class name
- **\$action** (*string*) – The action
- **\$fn** (*Function*) – Callback

Example:

```
gila::action('blog', 'topics', function(){ blog::tagsAction(); });
```

before (*\$c, \$action, \$fn*)

(static) Registers a function to run before the function of a specific action.

Parameters

- **\$c** (*string*) – The controller’s class name
- **\$action** (*string*) – The action
- **\$fn** (*Function*) – Callback

Example:

```
gila::action('blog', 'topics', function(){ blog::tagsAction(); });
```

addLang (*\$path*)

(static) Adds language translations from a json file.

Parameters **\$path** (*string*) – Path to the folder/prefix of language json files

Example:

```
gila::addLang('mypackages/lang/');
```

widgets (*\$list*)

(static) Register new widgets.

Parameters **\$list** (*Array*) –

Example:

```
gila::widgets ( [ 'wdg' => 'my_package/widgets/wdg' ] );
```

content (*\$key, \$path*)

(static) Register new content type.

:param String \$key Name of content type :param String \$path Path to the table file

Example:

```
gila::content( 'mytable', 'package_name/content/mytable.php' );
```

contentInit (*\$key*, *\$init*)

(static) Make changes on an existing content type.

Parameters

- **\$key** (*String*) – Name of content type
- **\$init** (*Function*) – Function to run when initializes the content type object

Example:

```
gila::contentInit( 'mytable', function(&$table){
    // unlist a column from content administration
    &$table['fields']['column1']['list'] = false;
} );
```

packages (*\$list*)

(static) Returns an array with the active packages names.

amenu (*\$key*, *\$item*)

(static) Add new elements on administration menu.

Parameters

- **\$key** (*Array*) – Index name
- **\$item** (*Array*) –

Example:

```
gila::amenu('item', ['Item','controller/action','icon'=>'item-icon']);
```

amenu_child (*\$key*, *\$item*)

(static) Add a child element on administration menu.

Parameters

- **\$key** (*string*) – Index of the parent item.
- **\$item** (*Array*) –

Example:

```
gila::amenu_child('item', ['Child Item','controller/action','icon'=>'item-icon
↳']);
```

config (*\$key*, *\$value = null*)

(static) Sets or gets the value of configuration element.

Parameters

- **\$key** (*string*) – Index of the element.
- **\$value** (*) – (optional) The value.

Returns The value if parameter \$value is not sent.**setConfig** (*\$key*, *\$value=""*)

(static) Sets the value of configuration element.

Parameters

- **\$key** (*string*) – Index of the element.

- **\$value** (*) – The value to set.

updateConfigFile ()

(static) Updates the config.php file.

equal (\$v1, \$v2)

(static) Checks if two values are set and have the same value.

Parameters

- **\$v1** (*) – First value.
- **\$v2** (*) – Second value.

Returns True or false.

hash (\$pass)

(static) Generates a hash password from a string.

Parameters **\$pass** (*string*) – The string to be hashed.

Returns Hashed password.

option (\$option, \$default=“”)

(static) Gets an option value.

Parameters

- **\$option** (*string*) – Option name.
- **\$default** (*string*) – (optional) The value to return if there option has not saved value.

Returns The option value.

setOption (\$option, \$value=“”)

(static) Sets an option value.

Parameters

- **\$option** (*string*) – Option name.
- **\$default** (*string*) – The value to set.

hasPrivilege (\$pri)

(static) Checks if logged in user has at least one of the required privileges.

Parameters **\$pri** (*string/Array*) – The privilege(s) to check.

Returns True or false.

dir (\$path)

(static) Creates the folder if does not exist and return the path.

Parameters **\$path** (*string*) – Folder path.

Returns string

make_url (\$c, \$action=“, \$args=[])

(static) Generates a url.

Parameters

- **\$c** (*string*) – The controller.
- **\$action** (*string*) – The action.
- **\$args** (*Array*) – The parameters in array.

Returns The full url path to print.

Examples:

```
$url1 = gila::make_url('blog','post',[1]);`` returns mysite.com/blog/post/1
$url1 = gila::make_url('blog','',['page1']);`` returns mysite.com/blog/page1
```

mt (*\$arg*)

(static) Returns modification times in seconds.

Parameters *\$arg* (*string/Array*) – Names of keys.

Returns *string/Array*

Example:

```
gila::mt('my-table')
```

updateMt (*\$arg*)

(static) Updates modification time in seconds. You can use this function from your model classes. The *cm* controller runs *updateMt()* for any content type in update action.

Parameters *\$arg* (*string/Array*) – Names of keys.

Returns *string/Array*

Example:

```
gila::updateMt('my-table')
```

10.2 Class event

class event

Registers and fires events (hooks)

listen (*\$event*, *\$handler*)

(static) Sets a new function to run when an event is triggered later.

Parameters

- **\$event** (*string*) – The event name.
- **\$handler** (*function*) – The function to call.

fire (*string \$event* [, *Array \$params*])

(static) Fires an event and calls all handling functions.

Parameters

- **\$event** (*string*) – The event name.
- **\$params** (*function*) – (optional) Parameters to send to handlers.

10.3 Class view

class view

Have methods that outputs the HTML

set (*\$param*, *\$value*)

(static) Sets a parameter from a controller action that can be used later from a view file.

Parameters

- **\$param** (*string*) – The parameter name.
- **\$handler** (*any*) – The value.

meta (*\$meta*, *\$value*)

(static) Sets a meta value that is printed later from `view::head()`.

Parameters

- **\$meta** (*string*) – The meta name.
- **\$value** (*string*) – The value.

stylesheet (*\$href*)

(static) Adds a new stylesheet link that is printed later from `view::head()`.

Parameters **\$href** (*string*) – The href attribute from the link.

script (*\$script*, *\$prop* = "")

(static) Adds a new script to be included in the output HTML.

Parameters

- **\$script** (*string*) – The src attribute from the script.
- **\$prop** (*string*) – Optional. A property for the script.

getThemePath ()

(static) Returns the path of the current theme.

head (*\$meta* = [])

(static) Prints all the head information in `<head>` tag.

Parameters **\$file** (*Array*) – (optional) Meta values to be printed.

getViewFile (*\$file*, *\$package* = 'core')

(static) Returns the path of a file inside theme or package folder.

Parameters

- **\$file** (*Array*) – The file path.
- **\$package** (*string*) – (optional) The package folder where the file is located if is not found in theme folder.

Returns False if file is not found.

render (*\$file*, *\$package* = 'core')

(static) Prints the view file adding the header.php and footer.php from theme.

Parameters

- **\$file** (*string*) – The file path.
- **\$package** (*string*) – (optional) The package folder where the file is located if is not found in theme folder.

renderAdmin (*\$file*, *\$package* = 'core')

(static) Prints the view file adding the admin/header.php and admin/footer.php from theme.

Parameters

- **\$file** (*string*) – The file path.

- **\$package** (*string*) – (optional) The package folder where the file is located if is not found in theme folder.

renderFile (*\$file*, *\$package = 'core'*)

(static) Prints the view file alone from theme.

Parameters

- **\$file** (*string*) – The file path.
- **\$package** (*string*) – (optional) The package folder where the file is located if is not found in theme folder.

includeFile (*\$file*, *\$package = 'core'*)

(static) Includes the view file without passing the.

Parameters

- **\$file** (*string*) – The file path.
- **\$package** (*string*) – (optional) The package folder where the file is located if is not found in theme folder.

menu (*\$menu='mainmenu'*, *\$tpl='tpl/menu.php'*)

(static) Displays a menu.

Parameters

- **\$menu** (*string*) – Optional. Name of the menu.
- **\$tpl** (*string*) – Optional. The view template to generate html.

widget_area (*\$area*, *\$div=true*)

(static) Prints the widgets of a specific area.

Parameters

- **\$area** (*string*) – The widget area name.
- **\$div** (*bool*) – (optional) Also print or not the widget inside a <div> tag with its title.

thumb (*\$area*, *\$prefix*, *\$max=180*)

(static) Returns the path of a thumbnail image of specified dimensions. If thumbnail does not exist it will create one.

Parameters

- **\$src** (*string*) – The path of original image.
- **\$prefix** (*string*) – The prefix name of the thumbnail.
- **\$max** (*int*) – (optional) The maximum width or height of thumbnail in pixels.

thumb_stack (*\$src_array*, *\$file*, *\$max=180*)

(static) Returns the path of a stacked image. If image does not exist it will be created on the fly.

Parameters

- **\$src_array** (*Array*) – The images to stack.
- **\$file** (*string*) – The name of the stucked image. It must have png extension.
- **\$max** (*int*) – (optional) The maximum width or height of thumbnails in pixels.

Returns The path to revisioned stucked image and the list of stucked photos.

Example:

```

$img = ["image1.png", "image2.png"];
list($file, $stacked) = view::thumb_stack($img, "tmp/stacked_file.png", 80);

/* Returned values

$file: tmp/stacked_file.png?12

$stacked[0]: ["src"=>"image1.png", "src_width"=>200, "src_height"=>150, "width"=>
↳80, "height"=>60, "type"=>2, "top"=>0]

$stacked[1]: false (2nd image was not stacked)
*/

```

10.4 Class db

class db

Class db prepare statements for mysql queries to the connected database. We use the global `$db` instance to access its methods.

query (*\$q*, *\$args*)

Runs a query and returns the result.

Parameters

- **\$q** (*string*) – The query.
- **\$args** (*array*) – Optional. Values to prepare the statement.

Examples

```

$result1 = $db->query("SELECT title,author FROM post;");
$result2 = $db->query("SELECT title,author FROM post WHERE user_id=?;",
↳[session::user_id()]);

```

get (*\$q*, *\$args*)

Runs a query and returns the results as an array.

Parameters

- **\$q** (*string*) – The query.
- **\$args** (*array*) – Optional. Values to prepare the statement.

Example

```

$result = $db->get("SELECT title,author FROM post;");
// Returns
[
  0=>[0=>'Lorem ipsum', 'title'=>'Lorem ipsum', 1=>'John', 'author'=>'John
↳'],
  1=>[0=>'Duis aute irure', 'title'=>'Duis aute irure', 1=>'John', 'author
↳'=>'John'],
]

```

gen (*\$q*, *\$args*)

Runs a query and returns a generator that yields the rows.

Parameters

- **\$q** (*string*) – The query.
- **\$args** (*array*) – Optional. Values to prepare the statement.

Example

```
$generator = $db->gen("SELECT title,author FROM post;");
```

getRows (*\$q*, *\$args*)

Runs a query and returns the results as an array. With rows fetched with `mysqli_fetch_row()`.

Parameters

- **\$q** (*string*) – The query.
- **\$args** (*array*) – Optional. Values to prepare the statement.

Example

```
$result = $db->get("SELECT title,author FROM post;");
// Returns
[
    0=>[0=>'Lorem ipsum',1=>'John'],
    1=>[0=>'Duis aute irure',1=>'John'],
]
```

getList (*\$q*, *\$args*)

Runs a query and returns an array with the values of the first columns from the results.

param string \$q The query.

param (array) \$args Optional. Values to prepare the statement.

Example

```
$titles = $db->get("SELECT title,author FROM post;");
// Returns
[0=>'Lorem ipsum', 1=>'Duis aute irure']
```

value (*\$q*, *\$args*)

Runs a query and returns the value of the first column of the first row of the results.

param string \$q The query.

param (array) \$args Optional. Values to prepare the statement.

Example

```
$res = $db->get("SELECT title FROM post WHERE id=1;");
// returns
'Lorem ipsum'
```

error ()

Return an error if exists from the last query executed.

Example

```
$res = $db->get("SELECT title,author FROM post;");
if ($error = $db->error()) {
    trigger_error($error);
}
```

close()

Closes the connection to the database.

Example `$db->close();`

CHAPTER 11

Indices and tables

- `genindex`

A

action() (gila method), 38
addLang() (gila method), 38
amenu() (gila method), 39
amenu_child() (gila method), 39

B

before() (gila method), 38

C

close() (db method), 45
config() (gila method), 39
content() (gila method), 38
contentInit() (gila method), 39
controller() (gila method), 37

D

db (built-in class), 44
dir() (gila method), 40

E

equal() (gila method), 40
error() (db method), 45
event (built-in class), 41

F

fire() (event method), 41

G

gen() (db method), 44
get() (db method), 44
getList() (db method), 45
getRows() (db method), 45
getThemePath() (view method), 42
getViewFile() (view method), 42
gila (built-in class), 37

H

hash() (gila method), 40

hasPrivilege() (gila method), 40
head() (view method), 42

I

includeFile() (view method), 43

L

listen() (event method), 41

M

make_url() (gila method), 40
menu() (view method), 43
meta() (view method), 42
mt() (gila method), 41

O

onController() (gila method), 37
option() (gila method), 40

P

packages() (gila method), 39

Q

query() (db method), 44

R

render() (view method), 42
renderAdmin() (view method), 42
renderFile() (view method), 43
route() (gila method), 37

S

script() (view method), 42
set() (view method), 41
setConfig() (gila method), 39
setOption() (gila method), 40
stylesheet() (view method), 42

T

thumb() (view method), 43

thumb_stack() (view method), 43

U

updateConfigFile() (gila method), 40

updateMt() (gila method), 41

V

value() (db method), 45

view (built-in class), 41

W

widget_area() (view method), 43

widgets() (gila method), 38