micro-MVC

Version 4.3.0

About

micro-MVC is an agile, small, productive and robust MVC framework for PHP with integrated AJAX support. micro-MVC is simple, yet agile and powerful, MVC framework and AJAX platform. Developers can write MVC-based and AJAX-based code rapidly and securely.

Although there are tons of free and open source MVC frameworks out there, they usually end up becoming too complicated as they are stuffed with a bunch of infinite extensions that only 5% of developers really use at some point. micro-MVC is trying to be hassle free, straightforward and helps you stay on track. micro-MVC API is robust and solves roughly 95% of all everyday issues you face under development with PHP and JS.

micro-MVC is size-optimized, secure, lightning fast and consumes the least resources possible. The framework runs on nearly any platform including embedded systems and has been tested extensively in several low-end and high-end hardware configurations. Also, it comes with out-of-the-box configurations for Apache and NGINX offering enterprise-ready security, smart caching and compression.

The latest release comes with the "Awesome!" platform that acts as a shorthand for the micro-MVC folder structure and provides an easy way for developers to access and manage their front-end and back-end code separately and the designers to work without conflicts almost in a parallel way.

Finally, on Dec-10-2020, micro-MVC has been audited, tested thoroughly for security vulnerabilities and put under stress with thousands of requests per minute from all over the globe. The results proved that micro-MVC is dependable and highly secure.

micro-MVC is being supported by **PROBOTEK** and George Delaportas.

Principles

micro-MVC is based on certain principles. The principles are of two kinds. Business and technical principles. The author of micro-MVC is an Enterprise Architect and as such does not see this project as just “yet another programming library or tool”. micro-MVC is a framework that takes under consideration the development process both from a business perspective and a technical point of view. Thus, considering all implications and aspects of both “worlds”, as a consequence,

the resulting software is following another path and paradigms, towards a more robust and business-centric implementation.

The principles that micro-MVC follows are:

1. KISS (Keep It Simple Stupid)
2. Separation of business, technical and UI logic
3. Stack-based architecture design pattern
4. Modular design
5. Easy maintenance
6. Small core libraries and tools
7. Best practices for code development (use of design patterns)
8. Custom code for full control and optimization per case
9. Use of extensions for complex stuff
10. Agile development for small to large teams
11. Integrated and extensible configuration files
12. Open architecture that supports any future or custom code
13. Well-tested and ready-made (out-of-the-box) tools and libraries
14. Security by design
15. Open source

Folder structure

The structure of micro-MVC is laid out like this:

micro-MVC	
|__ Awesome!/	*Awesome! platform*
| |__ code/	*All related linked folders for front-end and back-end developers*
| |__ design/	*All related linked folders for designers*
|__ Documentation/	*Documentation of micro-MVC*
| |__ Code/	*Code documentation*
| |__ Architecture/	*Architecture documentation*
|__ framework/	*The framework folder contains back-end code and configurations*
| |__ config/	*Configuration files and folders*
| |__ content/	*Content divided by language code (en, gr, it, ...)*
| |__ errors/	*Error web pages*
| |__ extensions/	*PHP and JS extensions divided in separate folders*
| |__ libs/	*Core code of micro-MVC*
| |__ logs/	*Typical web logs (error.log, info.log)*
| |__ misc/	*Miscellaneous code for routing, dispatching and requests security*
| |__ mvc/	*MVC configuration (folders, files)*
| |__ templates/	*HTML templates with PHP*
| |__ micro_mvc.php	*micro-MVC wrapper file*

|__ site/	*The site folder contains all front-end contents*
| |__ css/	*CSS files*
| |__ js/	*JS files*
| |__ php/	*PHP files*
| |__ pix/	*Picture files*
| |__ sections/	*HTML sections (header, body, footer, ...)*
| |__ index.phtml	*Main index HTML file*
|__ .htaccess	*Apache configuration file*
|__ nginx.conf	*NGINX configuration file*
|__ index.php	*Main entry point (bootstrapping) of framework*
|__ NOTICE.php	*Notice of license file*
|__ js_compactor.php	*JS Compactor – deployment utility*
|__ Awesome!.sh	*Awesome! platform executable shell file (use only on Linux)*
|__ Installation.txt	*Installation guidelines file*
|__ LICENCE.txt	*License file*

The architecture might seem confusing and complex in the beginning, but it is in reality very well organized and aligned with certain paradigms for proper code and design management. However, to decomplex developers and designers, micro-MVC offers an intuitive way to manage the framework. The "Awesome!" platform acts as a shorthand to the micro-MVC folder structure by significantly improving development speed, efficiency and confidence.

Configuration files

The configuration files of micro-MVC are under the "/framework/config" directory. The full structure of configurations directory is laid out like this:

config/	*Configuration files and folders*
|__ registry/	*Registry folder (JSON configuration files for PHP and JS extensions)*
| |__ php.json	*PHP extensions configuration type (core / user)*
| |__ js.json	*JS extensions configuration type (core / user)*
|__ misc/	*Miscellaneous folder (JSON configuration files for PHP and JS extensions)*
| |__ ext_autoload.json	*Configuration file for autoloading PHP and JS extensions*
|__ db.cfg	*DB configuration*
|__ gates.cfg	*Gates configuration (permitted "gate" codes for AJAX requests)*
|__ langs.cfg	*Languages configuration (available / valid languages)*
|__ params.cfg	*Parameters configuration (allowed URL parameters)*
|__ routes.cfg	*Routes configuration (allowed / enabled MVC routes)*

The structure of the config folder is simple. All configuration files have distinct and related names to their purpose of use. However, further information regarding *registry folder*, *gates.cfg* and *params.cfg* will be given to avoid any mistakes.

Under the registry folder there are two JSON files that “describe” which PHP and JS extensions are necessary for the framework to function properly. The options for each extension are: “core” and “user”. The corresponding extensions, under their PHP or JS folders, are inside a “core” or “user” folder respectively. The developer is free to delete any extension under user folders but not under core folders. This makes choices risk-free, provides reassurance and assists in a less confusing development process.

The gates.cfg file is part of a very special feature / mechanism of micro-MVC. All AJAX requests in micro-MVC pass through a validation system that acts mostly as a mini firewall and IPS. In order for any AJAX request to be successful, it has to know the name of a preconfigured gate, call its name and request access to execute some action on the server. All not known requests get blocked and no further action is taken. Finally, all cross-site AJAX requests are blocked from PHP and the Apache / NGINX configuration files included in the framework.

The params.cfg is a very handy configuration file that enables developers to allow certain parameters in the end of an MVC / SEO-ready URL like:
https://www.mysite.com/en/projects/?fbclid=IwAR22pzemuHtzBUO...
The “fbclid” parameter has already been added in the params.cfg for your convenience as it is a very popular parameter from Facebook.

All “.cfg” files contain configurations entries as plain text in English which are separated by a simple comma (“,”).

As you can see, micro-MVC offers a simple and secure mechanism for fine control over processes and files, useful to professionals who want to feel confident of their actions under complex and large enterprise projects.

Extensions

All micro-MVC extensions are under the “/framework/extensions” directory. The full structure of extensions directory is laid out like this:

```
extensions/          PHP and JS extensions divided in separate folders
|__ php/             PHP extensions
|      |__ core/     Core extensions
|      |__ user/     User extensions
|__ js/              JS extensions
|      |__ core/     Core extensions
|      |__ user/     User extensions
```

All PHP and JS extensions under core folders are necessary for the framework to function properly and should not be deleted or altered unless the developer really understands the micro-MVC framework. On the other hand, as it was mentioned before, the developer is free to

delete or alter any extension under user folders. This distinction assists the developer by introducing a risk-free choice under the development process and provides reassurance for a less confusing management.

Awesome! platform

Awesome! platform is truly awesome! In fact, the "Awesome!" platform is nothing more but a smart shorthand folder structure on top of micro-MVC that assists web development by separating code from design. This provides a concrete and intuitive platform for front-end, back-end programmers and designers.

The Awesome! platform files are under the "/Awesome!" directory. The full structure of this directory is laid out like this:

Awesome!/	*Awesome! platform*
|__ code/	*All related linked folders for front-end and back-end developers*
| |__ front-end/	*All related linked folders for front-end development*
| |__ back-end/	*All related linked folders for back-end development*
| |__ dispatching/	*All related linked folders for dispatching actions (MVC / AJAX)*
|__ design/	*All related linked folders for designers*

Essentially, sub-folders of Awesome!, contain smart links to real micro-MVC folders. If the programming environment is Windows, the developer has direct access to the folders. If the programming environment is Linux the developer should execute "Awesome!.sh" in the root directory of micro-MVC. The shell script will delete all Windows links under the Awesome! folder and will replace them with the proper Linux links. Then, access to the folders is possible.

Miscellaneous features

micro-MVC further enhances and extends certain programming paradigms and design patterns. It also introduces easier ways to handle code and requirements. The following list refers briefly on those paradigms and describes the reasoning behind the unique / new way of handling them:

- Virtual MVC routes

All MVC routes in micro-MVC are virtual. This means that there is no need to have an actual path and filename to reflect a view and a model on the file system. Then, the developer can simply echo HTML in the virtual view. Also, if indeed a route exists as a physical view file (.phtml), there is no need for a model file (.php) to exist. This greatly increases productivity as developers might just need an MVC route to show stuff but not necessarily to utilize a model to load data. Finally, MVC routes that reflect on real files are under the "/framework/mvc" directory separated as views and models.

- Reuse of open DB connections

There is usually a concern about how many persistent DB connections a server can handle under stress. For that purpose, the core of micro-MVC provides a way to store, restore and reuse

already initialized DB connections. If there is an initialized connection all executed SQL commands use that connection and therefore significant memory resources are being saved.

- Extension re-inclusion protection

Cases like dependency re-inclusion, where a developer reloads again and again an extension and breaks the code or makes the program behave abnormally, are solved by design.

- Server-side controls (dynamic and pre-formatted HTML elements)

micro-MVC includes a very powerful PHP extension. This extension is Splash (spl@sh). Splash is a unique library that empowers PHP developers to use server-side controls like the ones that ASPX developers use in .NET. Splash formalizes both HTML elements and methods so that any control prints and functions the same way on any browser and any operating system. Splash is well-documented and you can find more information in the “splash” extension folder.

- JS Compactor

micro-MVC provides tools to assist developers deploy their work easier and faster. The JS Compactor is a simple utility that minifies, merges and compacts all JS extensions in one file. You may run the “js_compactor.php” directly from the root directory. The compactor will create a new file under the “/site/js” folder. Then, you may unload all your autoloaded or manually loaded JS extensions and remove the comment in the head of “/site/index.phtml” file, as seen in the code snippet below.

```
<!-- <script src="/site/js/all_ext_min.js"></script> -->
```

Best regards,
George Delaportas
Enterprise Architect & IT Security Expert
Code Hacker
https://github.com/g0d/micro-MVC