XBox Game Voting Application

Completed: Jan 3rd, 2013
Document Version: 1.0.0, Jan 3rd, 2013, very early in the morning (3am)

Purpose of this document

The purpose of this document is to highlight the code and what it attempts to accomplish. All requirements for the application are in the document sent by Nerdery and I will try not to be redundant in explaining those requirements while going over the code.

Framework Architecture

The framework used to complete this application is a custom-built “light” MVC framework. What this means is that it is very light-weight and meant only for small projects. This framework was just recently built (just for this project) and I cannot determine at this time if it would be good for larger projects. I will say that I’ve tried my best to make the framework as easy as possible to understand and extend.

Core Features

- Namespaces
    - Namespaces are used in order to keep from colliding with other classes of the same name.
- Autoloading
    - We get rid of most of those pesky include and require functions using autoloading from PHP.
    - Use the XBoxApp\Framework namespace structure to make use of this feature when extending the framework.
- Theming
    - A basic theme setup is available that works much like Wordpress in an Object-Oriented way.
    - Your controller works off the main theme class <theme>.php and uses the RouteProvider class.
    - ThemeProvider allows you to render partials in your theme and will automatically include the header and footer in that partial.
    - Segments using ob_start and ob_end is not yet supported... I am kinda lame.
- Models
    - All models are in the XBoxApp\Framework\Models namespace.
    - Models handle the game adding/voting functionality as well as the user validation.
        - GameModel - game adding/voting functionality
        - UserModel - user validation
        - ServiceApi - Singleton model for accessing the web service
- SQLite3 Database
    - While we are not “caching” the results an “offline” mode was created due to certain circumstances which prevented me from accessing the internet for

about half the time of this project. When “offline” mode is turned off the SQLite3 database is used merely as a way to sort the data.
    - Sync - A sync function was created to ensure that at all times data is coming from the Nerdery Web Service and nothing is cached. Essentially this is no more useful than holding the data in an array and doing a usort when offline mode is turned off.
- RouteProvider
    - Useful way to create and control (controller) the routes in your framework.
    - When a route isn’t available it sends the user to a 404 page which can be customized in the theme.
- Config
    - config.php provides options to set the theme, turn off “debug” mode, set whether or not we are running in “offline” mode, and (for testing purposes) allow us to completely ignore the user voting validation parameter.

MVC Framework Idea

The idea behind this framework was to take concepts from Wordpress, CodeIgniter, Laravel, and some others that I can’t think of at the moment. Everything starts with the ***\XBoxApp\Application*** class and the ***startup()*** function. This begins the process of registering the autoloader function, setting the theme, and discovering the route.

After this is started all functionality of the application comes from the models and the <theme>.php “controller” file. This is equivalent to the functions.php file in Wordpress. For each route (non-ajax) you can use the ***ThemeProvider::theme()->render()*** function to render the current page. This will ensure that you are rendering using the header.php and footer.php files. Another alternative would be to use the ***\XBoxApp\Framework\View::render()*** function which is useful when rendering partials using AJAX.

I will repeat, this is a basic implementation of a MVC approach that is inspired by frameworks such as AngularJS, CanJS, Laravel, Wordpress, etc. I found it useful for display my understanding of “concepts” in PHP which is why I decided to take this approach.

Understanding the Code

I’ve tried to document as much as possible inside the code. All class methods use a camel-case approach rather than the usual ‘_’ approach used by many of my fellow PHP developers. I don’t really have a preference, I can live with either approach.

I have used a singleton pattern in parts I felt may have made the code look more elegant. I try not to be too “clever” with my code so that people can understand immediately what’s going on, however I do feel that singletons have their place in this world and should be used when necessary.

Conclusion

I doubt this document answers all of your questions about the code; my intention was to help give the person reading this document a running start before they dive into the code. I do love to put comments in my code so feel free to laugh, cry, scratch your head, or share the feeling that I do which is “code is an art form that we’re constantly trying to master”!