BuckyBall

The general purpose framework

The original feature set target was based on Magento's base framework, but eventually it has surpassed the flexibility.

Issues with Magento framework from developer standpoint:

IDE integration

Autocompletion

Generic class factory interferes with IDE auto-completion/method reference:

```
// Magento:
$data = Mage::getSingleton('core/session')->getData();

// BuckyBall:
$data = BSession::i()->data();
```

File naming

If you have multiple files open in a regular IDE, it is very hard to know which file is which just by the name (ex: Abstract.php, Source.php, Observer.php)

BuckyBall: 1. The framework itself is only 1 file, other modules are encouraged to bundle classes into as less files as makes sense.

Directory structure

Magento has very deep level directory structure.

Buckyball does not force a specific application or module folder structure, and encourages to have it as simple as possible.

Debugging

- Backtrace stack in Magento is unnecessarily huge.
    - BuckyBall is much much lighter, with more meaningful steps.
- Magento raw objects are impossible to var dump, have to use →debug() method.
    - BuckyBall addresses this by encouraging simple objects without circular references.

- Magento doesn't track which configuration came from which module.
    - BuckyBall records each hook into request workflow.

Module structure

Folders

Magento has hard enforced folder structure, which requires modules to be split into multiple deep folders, creating issues with module management and deployment.

BuckyBall restricts all module files to be in the same folder.

Customization

Class/method override

Magento allows overriding a class only once, creating compatibility issues between multiple modules or customizations that require overriding same or different methods within the same class.

BuckyBall allows overriding each method separately. There's also a functionality to augment a method, which can be done by multiple modules over the same method.

```
// override a class, each new override will cancel previous (the module of
last override is recorded)
BClassRegistry::i()->override('Old_Class', 'New_Class');

// override a method, each new override will cancel previous
// $callback is PHP callback, example:
// function callbackMethod($origObject, $arg1, $arg2) { }
BClassRegistry::i()->overrideMethod('Class', 'origMethod', $callback);

// augment a method, each new augmentation will use result of previous
augmentation
// function callbackMethod($result, $origObject, $arg1, $arg2) { }
BClassRegistry::i()->augmentMethod('Class', 'origMethod', $callback);
```

HTML override

Sometimes for a small HTML change Magento requires overriding a whole phtml template.

BuckyBall includes phpQuery plugin, which allows jQuery style HTML changes on the server.

Adding existing apps/libraries as modules

Because BuckyBall does not mandate a module structure, any library can be created as a module. The

only requirements are: manifest.json to declare a method and bootstrap callback method.

System load

Due to large number of files, deep backtrace and object structure, Magento tend to be very heavy on system resources. To solve some performance issues, extensive caching is used.

One of BuckyBall's main goals is to be light on the system. Caching is very optional.

Planned applications

CMS

The CMS will provide facility for structure of the site and data versioning of any model that would use the functionality (including other apps, such as eCommerce)

eCommerce

The goal is to create an easy to install, maintain, extend and customize eCommerce suite, which would provide most of the functionality of Magento, while solving inherent Magento framework issues (memory, speed, complications, number of files/tables)

Deployment

Get is here: http://github.com/unirgy/buckyball

The framework itself is `bucky/framework.php`, useful plugins: `bucky/plugins`

Recommended app structure

```
index.php                   - Main bootstrap file
.htaccess
bucky/
    .htaccess               - deny from all
    framework.php           - Main framework
    lib/                    - BuckyBall libraries
    plugins/
        <base plugins>
lib/
    <3rd party libraries>
modules/                    - downloadable and custom modules
    cms/
        <cms modules>
```

```
    shop/
        <ecommerce modules>
    app/                        - local custom application
        <local app modules>
storage/                        - this folder is writable for web service
(recursively)
    protected/                  - all subfolders should be inaccessible from
web
        .htaccess               - deny from all
        cache/                  - if any module uses cache, store here
        config/                 - default storage for configuration files
        sessions/               - web sessions
    media/                      - web accessible media assets (product images,
, etc)
        <media files>
```

External resources, such as scripts, styles and theme images are to be contained within community modules or local app modules.