

Must haves

Everything in this section is stuff we absolutely must do no matter what.

Performance improvements

The things in this section are being done primarily for performance reasons, even if there are other benefits.

Replace Slaves with Multi-Master

Effort: 24 hours

Requirements:

- Remove all code and configs for slave database queries
- Add configs for multi-master connections that look like this:

```
“db” : [  
  {  
    “dsn” : “DBI:mysql:www_example_com;host=localhost”,  
    “username” : “dbuser”,  
    “password” : “dbpass”  
  },  
  {  
    “dsn” : “DBI:mysql:www_example_com;host=remotedb.example.com”,  
    “username” : “dbuser”,  
    “password” : “dbpass”  
  }  
],
```
- Migrate existing dsn, dbuser, dbpass directives to use the new config format.
- Migrate existing failover directive to use the new config format
- Change the database connection code in WebGUI::Session to try to connect to the first entry in the list, and then the second, and so on for automatic failover

Problems Solved:

- The current slave set up does very little if any good due to the way WebGUI uses databases
- This allows each web node to have a configured list of databases, each with a different master database which will distribute the load nicely, and provide failover

Replace existing cache system with Memcached

Effort: 88 hours

Requirements:

- Add memcached to WRE
- Trash existing cache class and drivers

- Write a class for using memcached in the best way possible
- Update all uses of cache through-out WebGUI

Problems Solved:

- Removes plugability, which removes a layer, which improves speed
- Takes load off DB
- Allows larger scalability by moving caching to additional servers

Eliminate pluggable templating and replace with Template Toolkit

Effort: 400 hours

Requirements:

- No more periods in template variable names
- All existing HTML::Template templates must be migrated 100%
- All existing HTML::Template::Expr templates must be migrated on a best effort basis
- All existing Template Toolkit templates must be migrated to use any new variable names.

Problems Solved:

- Removes plugability, which removes a layer, which improves speed
- Improves upgradability because we'll only have one template engine to migrate templates for
- We can take heavy processing variables and turn them into functions that only get generated when needed, which will speed up rendering
- Only one thing to learn for templaters
- More flexible templates
- Less code generating template variables

Subclass Template to create Style

Effort: 16 hours

Requirements:

- Subclass template to create style
- Move style wizard into style subclass
- Add footer field
- Make javascripts output to footer field
- Hard code namespace to be "style"

Problems Solved:

- Pages will render faster if javascripts our output last per YSLOW guidelines
- Style wizard won't be hacked into template anymore
- We can write anything custom for styles that we need, because they're a different class of templates

Load YUI from CDN

Effort: 8 hours

Requirements:

- Add a field to the config file called `yuiUrl` and another called `yuiHttpsUrl`.
- Add methods to `WebGUI::Session::Style` to handle adding YUI URLs to the head/foot blocks of the style

Problems Solved:

- YUI is a large part of our page view, and by moving it out to a CDN we can increase the performance of our pages dramatically

New admin console

Effort: 240 hours

Requirements:

- Write a brand new admin console framework
- Rewrite admin bar to fit in the admin console framework, and to allow dynamic add/remove of both tabs and items
- Remove all references to the `AdminBar` macro
- Rewrite asset manager to load everything via ajax, and integrate it with admin console

Problems Solved:

- Speeds up the UI by not reloading the same components (asset manager, toolbars, admin bar) over and over again
- Gives us a new, cleaner, simpler UI

Add exceptions on Storage

Effort: 8 hours

Requirements:

- Add exceptions to storage
- Remove old error handling mechanism
- Update existing code that uses it

Problems Solved:

- We don't provide good feedback to the user when something bad happens on a file upload, exceptions will help with that

Eliminate ErrorHandler

Effort: 40 hours

Requirements:

- Eliminate “fatal” which kills stuff

- \$session->log becomes the way to interface with the logger
- Logger no longer has impact on whether the page continues execution

Problems Solved:

- A way to overcome fatals
- Logging and exception handling should be separate

Add expires headers for all asset output

Effort: 16 hours

Requirements:

- All assets should add an Expires HTTP header equal to whatever cache timeout settings they have.
- Not all assets have cache timeouts and some only have them for certain types of users. If an asset doesn't have a cache timeout then it shouldn't get an Expires header.
- Page Layout assets there is an implied cache timeout of 60 seconds for visitors.

Problems Solved:

- Improves page cachability, and our default YSLOW score

Add ETags to asset output

Effort: 16 hours

Requirements:

- All assets should add an ETag HTTP header which consists of User ID + Asset ID + Revision Date. The resulting ETag would look like: ETag: "3+AJKesselkael_esle-3sj3+12039477446"
- The Asset.pm content handler should be updated to handle the If-None-Match request as described here: <http://developer.yahoo.com/performance/rules.html#etags>

Problems Solved:

- Improves page cachability and our default YSLOW score

Robustness

The things in this section are being done primarily for robustness, even though there may be other benefits. For the sake of this document robustness is defined as eliminating duplicate code, or standardizing APIs to reduce code or code complexity. In other words, help produce less bugs, make the code more testable, and require less code to use the API.

Replace existing auth system entirely

Effort: 320 hours

Requirements:

- Build a new auth system that is easily subclassable and that actually makes sense
- Convert the WebGUI and LDAP auth methods to use it
- Most things including user interface should be inheritable, but overridable
- Allow for alternate login boxes to appear on the same screen, such as WebGUI/LDAP login plus Open ID
- If Open ID is still showing strong, make it a built in auth mechanism
- Build in a single sign on mechanism that can be used so that WebGUI sites can authenticate against each other

Problems Solved:

- Auth modules no longer take a week to write
- Auth modules can be subclassed
- Single sign on becomes possible

Improved asset state

Effort: 40 hours

Requirements:

- Make archived its own flag in the asset table and pull it out of state
- Add “new” state, which will be the state of an asset after it has been created, but before it has been given its initial properties
- The `www_add` method will create an asset and set its state to new, then instantiate that asset and call `www_edit` on it
- Remove the `newByHashRef` method
- `www_editSave` should set the state to pending
- A workflow activity should be created to delete assets with a state of new after they are six hours old
- Remove asset temp space and associated workflow

Problems Solved:

- Right now when you first create an asset it doesn't exist, so you can't attach anything to it (no ajax)
- This will allow save/preview operations on things like forum posts, and story manager preview, where the attached files actually get attached
- This will get rid of the need for asset temp space and the crazy crap we have to do with wiki pages because of it

Require rigid definition for assets

Effort: 64 hours

Requirements:

- `www_edit`, `www_editSave` and `getEditForm` are removed from `WebGUI::Asset` and moved into the admin console
- All fields are required to be 100% defined in the asset definition

- International labels are defined in the definition, but not evaluated there
- \$session is no longer passed in to definition

Problems Solved:

- Lowers asset footprint because we don't need all that extra code in there
- Allows an admin mode edit of an asset to be different from what the end user sees (on a forum post for example)
- Allows for robust checking of data types
- Allows for install/uninstall on all assets
- Allows for dynamic creation/update of asset tables
- Allows for better data integrity and testability
- Definitions can be processed more quickly if internationalization doesn't have to be evaluated there, plus if definition is evaluated multiple times in a single page view it's not re-evaluating internationalization each time
- definition() can be called at compile time to auto-generate many methods.

Clean up form system

Effort: 80 hours

Requirements:

- API for using WebGUI::HTMLForm and other form classes must remain as backward compatible as possible, slight alterations will be tolerated
- Eliminate all deprecated methods from form controls
- Require a rigid definition for all form controls, and refactor the existing definition because it's too loose and weak
- Form structure is built out like:


```
{
  properties => { hash of properties like method, enctype, action },
  fields => [ array of form control objects ],
}
```
- Form fields aren't rendered until the print() method is called
- Add a processFormPost() method which returns a hash reference of all the values of the form validated and defaulted properly. This would become the defacto way to process forms on the return.

Problems Solved:

- Eliminates a ton of duplicate code for processing forms from a form post
- Allows for more programatic use of the forms control system in crud sub systems like WebGUI::Crud and WebGUI::Asset

Replace WebGUI upgrade system

Effort: 90 hours

Requirements:

- Build a new upgrade system that is more robust and more modular than the current one
- Allow for a user defined maintenance screen and move the basic one out of the docs folder
- Move all upgrade files into a newly created var folder and out of the docs folder.
- Do upgrades from version to version for each site, rather than doing all upgrades for one site and moving on to the next
- Pull upgrades from git one rev at a time
- Upgrades are broken up into upgrade revs, upgrade units, and upgrade components
 - Upgrade revs are 8.0.1 to 8.0.2
 - Upgrade units are “added comments to article asset”
 - Upgrade components are “update template”, “apply database changes”, etc
- Upgrade units are ordered by number so they can occur in a particular order, and are just a folder within an upgrade rev
- Upgrade components are ordered by number and are a particular file type
- Upgrade component file types are:
 - sql – an sql file like mysqldump creates
 - api – uses the WebGUI API to manipulate data and or structures
 - delete – a text file containing a list of paths to be deleted
 - wgpkg – a webgui package that needs to be imported
 - module – a text file containing a perl module (or modules) that need to be installed
 - prereq – most likely a bash file which gets executed to tell if some prereq that needs to be present and configured to continue exists
 - pl – a perl file that will just arbitrarily be executed, that the upgrade system doesn't have any other knowledge of what it's doing
 - readme – a text file to display to the user and let them read before continuing
 - notice – a text file to display to the user through the usual output, but that doesn't pause the upgrade operation

Problems Solved:

- Users can add/remove components from the upgrade easily for things they've back ported or custom components they've created
- Eliminates the whole concept of API based stop points, so users can upgrade seamlessly from 8.0.1 to 8.9.4 without having to worry about stop points along the way
- Allows for stop points defined by users along the upgrade path if they are having problems with a particular upgrade they don't have to go all the way back to where they started
- Allows us to easily copy units and components from one upgrade to another for the purpose of shunts between betas and stables

Add asset helpers

Effort: 40 hours

Requirements:

- Replace asset toolbar and associated methods with asset helpers
- Create a pluggable API for adding helpers
- Helpers should be passed a session, and an asset object to be used as a start point
- Replace all existing toolbar functions with helpers: edit, delete, cut, copy, promote, demote, edit branch, view

Problems Solved:

- Right now there is no cohesion between the menu options in the asset manager or the menu options in the toolbar rendered in the inline view, which makes errors more likely when making changes
- People often want to add helpers for repetitive tasks, or have “edit branch” functionality for specific types of assets, and currently there's no way to do that

Internationalization

Effort: 80 hours

Requirements:

- Replace existing internationalization system with libintlperl

Problems Solved:

- Less memory used
- Less duplicated text for translators to translate
- Programmers put initial translation right into code, which means everything gets translated right away
- Faster than the current mechanism

Investigate and do the best we can

Everything in this section needs more investigation. If we can find a reasonable way to include these in WebGUI 8 in a reasonable amount of time, then we should get them done.

Unify APIs

Requirements:

- All objects in WebGUI should have write, create, update, delete, new, get, and getId methods
 - The write method is new, and would automatically be called by update() to write properties to the database, and would manually need to be called if using the regular setters
- All objects should have accessors and mutators for each property and they should take the form of get_propertyName and set_propertyName
- Accessors and mutators should be auto generated where possible.
- update() and get() should use the accessors and mutators for returning their values

Hurdles:

- This might take more time than we have
- There are certain complications with dynamically generating accessors and mutators on classes that don't have a rigid definition like Thingy records, profile fields, and possibly WebGUI::Crud classes

Problems Solved:

- Easier for developers to pick up new APIs
- Possible testing improvements
- Likely quality improvements (less exceptions)
- Code is easier to read/debug

Merge Settings with Config File

Requirements:

- Eliminate the settings table in favor of storing all settings in config file

Hurdles:

- Have to overcome the limitation that config files are only loaded at start up time, memcached could help with that

Problems Solved:

- One less hit to the database
- One less API to maintain
- Settings can be hierarchical

Replace WebGUI search with something

Requirements:

- Index and search meta data as well as key words
- Must support stemming
- Must still index in real time

Hurdles:

- Don't know what could work for this:
 - Sphinx might work, but it has a big time with real time indexing
 - C version of Lucene might work
 - KinoSearch is Perl, but may not be far enough along
 - Lucy has Perl, bindings but may not be far enough along
 - SWISH-e is a nice big search engine, but isn't flexible enough
 - Maybe we need to make a new hybrid of what we already have

Problems Solved:

- The current search isn't terribly accurate
- The current search isn't weighted at all
- The current search isn't capable of stemming
- The current search can't search non-keywords data
- The current search has to go away if we switch away from MyISAM tables
- The current search is built in-house so it's code we have to maintain

Merge assetData, and asset class tables

Requirements:

- Make assets responsible for constructing a table to store all their data which combines auxillary tables such a metadata, the class table (article, poll) , assetData, super class tables (wobject, sku), and any aspect tables into one table

Hurdles:

- If we get rid of the assetData table how do we search for all assets owned by a specific user or whatever else is in that table currently
- If we eliminate the assetData table we then have to create a url table, which shouldn't be versioned or it will cause performance problems
- If we can't eliminate the assetData table then we won't get as big of a performance boost due to the composite key join

Problems Solved:

- The whole asset system could get as much as a 10x increase in performance, but no matter what, will be sped up considerably
- Assets will be able to manage their own table structures like WebGUI::Crud which can make upgrades more reliable
- Less database table corruption, and less bottlenecks from sharing tables all over the place

Move lineage to the side

Requirements:

- Add a getChildren() method which returns an iterator and uses parent child relationships for lookups
 - It should have many of the same options as getLineage, for things like ignoring certain classes and whatnot
- Make getChildren the default way to look up most assets rather than getLineage
- Add a rank field to the asset table
- The lineage field would then truly become an index, and therefore could safely be rebuilt at any time
- Rebuild the rebuildLineage.pl script to not take in the entire lineage before it starts rebuilding, but to rebuild it one parent/child at a time

Hurdles:

- It's unknown how deeply this would impact the system and especially third party modules
- Rebuilding lineage would not fill rank gaps

Problems Solved:

- Because lineage would be strictly an index it could be rebuilt while the site is in operation
- Parent child lookups are faster when all you need are the children, and that's what you need most of the time
- Even if lineage has to be rebuilt, nothing would ever lose its order
- More complex keys can be set up with parent child relationships than with lineage, because lineage is nearly max key length by itself.

Investigate alternative databases

Requirements:

- Replace MySQL MyISAM with something with transactions and foreign keys

Hurdles:

- Don't know what database to use
 - MySQL with InnoDB is a top contender
 - MariaDB is worth a look
 - Drizzle could be really cool if its anywhere near ready
 - Percona MySQL is worth a look
 - OurDelta MySQL is worth a look
 - PostgreSQL may be worth a look, but it's so different from what we use currently that it would likely get killed in the mire of work it would create to switch to it
- There's going to be quite a bit of work to get webgui to play nice with a new DB

Problems Solved:

- Less database corruption due to ACID compliant transactions
- Foreign keys are faster for some types of joins

If we find enough time

Everything in this section is completely optional. If we somehow complete the huge list of items before this, and we find ourselves with some time, then we should do these things.

Replace WebGUI::Session::DateTime with WebGUI::DateTime

WebGUI::DateTime was meant as a replacement for WebGUI::Session::DateTime, but it never got all the compatibility stuff that it needed to fully replace it. We should find some time and work on that so we can be down to one way to do it, or at the very least so that WebGUI::Session::DateTime uses WebGUI::DateTime.

Finish WGFS

WGFS needs to be completed and integrated into the core. This will give us huge marketing advantages, and increase usability. However, it's probably about a 6-8 week effort by itself, so may not be possible to do.

Replace TinyMCE with YUI Rich Edit

TinyMCE is quite bloated and takes a long time to load. YUI has a nice rich editor, but it's missing some key features like being able to apply styles from a style sheet. We should work to replace TinyMCE with YUI Editor for both robustness and user interface performance.

Enhance RSS Aspect

Change the RSS aspect to allow for more than one feed coming from an asset. This is in the nice to have list because it could be achieved by adding to the API rather than refactoring the API.

Currently, the RssAspect plugin will only provide 1 RSS method per Asset, and with a fixed URL (?func=viewRss). Assets like the Gallery have multiple RSS feeds (listFilesForUsers, listAlbums, etc.)

The Aspect should be changed so that assets can specify what feeds they want, what the query fragment is, and what method is to be used for getting items in the feed. From that, the Aspect will:

- Generate fields in the edit form for each Feed (feedTitle, feedDescription, feedCopyright, et. al.). This will have a big impact on the RssAspect schema.
- Make the www method for that feed.
- Handle exporting all feeds.

- Add header information for all feeds.