Refactoring for Performance and Stability

This document contains a list of changes that will improve the performance and stability of Platform-X.

# Backend Changes

These items will not affect layout or presentation of the user interface, but will positively impact the performance of the overall application. Changes listed below will also improve the stability of the build by making the code less verbose and more flexible.

1. Eliminate the Bfw.Agilix.Service projects. These projects were originally intended to provide higher level functions on top of the base DLAP calls. However, after going down this path we now realize those services are far too one-to-one with DLAP to be useful. They actually increase the amount of code a developer has to write in order to support a new DLAP command.
2. Implement a new set of Bfw.Px.Biz.ServiceContracts that use the Bfw.Agilix.Commands wrappers directly. This will improve code clarity and allow for specific performance enhancements.
3. Implement full support for DLAP command batching in the Bfw.Agilix.Commands project. This will allow us to reduce the overall number of requests we make against the DLAP server. In this case performance will be improved by reducing network and other communication overhead.
4. Refactor the Bfw.PX.Notes library so that it no longer use LINQ-to-SQL. Instead, we will use the Dapper.NET datamapper. This will reduce code and deployment complexity as well as improving performance by up to a factor of 2.
5. Ensure that all backend code is wrapped in at least one profiler for performance monitoring
6. Implement handler so that logging is stored to a database asynchronously instead of to the system logs
7. Integrate the Mvc-Mini-Profiler by implementing a new ITraceManager. This will allow us to store and access performance profiling information more efficiently than the current method.
8. Perform a full code review on all Bfw.PX.Biz and Bfw.Agilix libraries to ensure that they are all coded as uniformly as possible and fully documented.
9. Run a documentation extraction tool like Doxygen and include it in the Continuous Integration environment.
10. Profile the commenting database and refactor the stored procedures and schema as necessary.
11. Implement a distributed cache manager and cache commonly accessed items into a long term cache (e.g. users, courses, enrollments, etc).
12. Using the distributed cache, store short-term objects like Agilix items and resources
13. Build a local file cache so that data files that are transferred server-to-server are cached for a short time locally. This will reduce the time it takes to load a content page significantly.
14. We are using resources to store item descriptions, necessitating a separate DLAP call when we need that information. We may not need to do so, and be able to just store descriptions on the item directly.
15. Where it makes sense to do so, convert app settings configuration to configuration sections.

# Frontend Changes

These items affect the layout and client-side behavior of one or more pages in Platform-X.

1. Refactor CSS and make use of the LESS CSS library. This will speed up development of CSS code and reduce the amount of CSS that needs to be written.
2. Use a resource loading tool like yepnope to only load the necessary scripts and CSS on each page. This reduced the overall footprint of each page.
3. Refactor existing javascript into jQuery plug-ins that communicate with each other through the redesigned PxPage object. This will increase the efficiency of our client side code and improve maintenance.
4. Integrate the use of JSLint and JSMin into our QA and production builds. This will further reduce our per page footprint in terms of data transfer.
5. Create a Static HTML reference site so that designers can more easily work on the per-course style changes (e.g. changing colors, graphics, etc). This will remove the need to create PDF comps for each product.
6. Eliminate any dead code, unused views, unused actions and controllers, etc
7. Move javascript files into folders based on their type rather than the page they are tied to. For instance all widgets should be in the widgets folder.
8. Extract the list of CSS classes used and apply a single, consistent set.
9. Refactor the markup as needed to ensure that it is a compact and flexible as possible.
10. Combine all graphics used for buttons, highlighting, etc into sprites to reduce network overhead and speed up page loading.
11. Refactor larger Javascript files into pieces. Specifically, large pieces of functionality, like quizzing, have their client side logic in one file, and this needs to be broken up.

# Specific Refactoring

1. Plan for the January Home Page Wizard included a lot of necessary refactoring and should continue as planned.
2. Assignment Center needs to be refactored so that it loads data more efficiently and it's client side code is easier to maintain.
3. The existing SSO stub application should be modified to more accurately reflect RA/Novell so that our local development is more closely related to the QA & PROD environments.
4. Work directly with Agilix to analyze our requirements for creating quizes, especially the process of creating question banks, in order to find a more efficient method for loading and searching through our large banks of questions.

# PX Reliance on RA/Novell

1. Relationship between RA/Novell and PX needs to be addressed such that PX no longer has code that redirects the user. All redirects based on authentication or access entitlement should occur in the RA/Novell layer before the request ever reaches PX.
2. All information from RA/Novell should be passed in as headers, currently there are several direct calls to RAWS to get around missing Novell functionality.