MashupJS ToDo

# Continuous Tasks – Perform after every complete task

* Continuously update the Mashup Style Guide
* Monthly – update all components
  + Use this to find global packages (grunt, gulp, bower) Npm list –g –depth=0
  + Use Bower to update client side libraries
  + Update NuGet packages
  + Update NPM packages
  + Check for tools updates (VS, powertools, web express, resharper, tortoise, visualsvn, node/io, etc)
  + Bootswatch updates that correspond with Bootstrap

# All Tasks

## One Off Tasks

* Move documentation to its own repository
  + Update the mashup readme.md to point/reference the new doc repository
  + Create new repo and copy documents to it.
  + Go through and update all image links. They will all break.
  + Remove old docs from MashupJS repo
  + Verify all blogs and doc file links work.
* Add Bootlint to grunt/gulp tasks: <https://github.com/twbs/bootlint>
* Script out database
* Add new Revealing Pattern where and object is returned to make code easier to read. Update style guide.
* Verify core uses good names. (Update style-guide)
* Update documentation on how to use cacheService. We pass in the url options now and the exact heartbeat url.
* Update sytle-guide with new directory structure.
  + Note: I moved out the app.js, index.html, custom css, etc.
* Update style-guide with style guides for C#, WebApi, CSS,
* Update Code and Syntax Highlighting in all existing markdown
* Add <https://highlightjs.org> to code examples.
* Add Angular Hint
* Find a place for Babel, the new 6to5, in the Mashup. Or see if I can do all I need with TypeScript.
* Mature my usage of promises. I know there are cleaner ways to do promises if I understood them more.
* Look at using CDNs for libraries we use. Or the use of CDN’s with local backup.
* Add async await to WebApi
* Make it possible to change your theme on the fly and save the theme selection to cache.
* JS References in VS
* Add and document how to add JavaScript library references in Visual Studio.
* Fix Browser Link – Is broken for some reason. Probably just out of date.
* Add Toaster notification
* Add ng-inspector to tools.md <http://ng-inspector.org/>
* Add names to anonymous function
  + Fn.displayName – might not work in all browsers
  + Add solution to style-guide
* Add feedback module so users can let us know when things aren’t going well or look wrong.
* Implement AngularLint
* Pretty URLs: <http://scotch.io/quick-tips/js/angular/pretty-urls-in-angularjs-removing-the-hashtag>

## Projects

**Create Mashup Implementations**

* **Mashupjs.AuthApi**
  + Retrieve AD credentials including groups
  + Retrieve custom application roles from SQL Server SPs
* Mashupjs.ExampleDataApi
* Create a big WebApi for Sample Data
* Document/Blog this and open up to open source.
* Examples with sync and async/await
* Example with Redis Cache and without
* Examples with save and queue to save
* Create a client to go with the WebApi to demonstrate
* Add some kind of timing/performance metrics
* **Mashupjs.lodashExamples**
* Mashupjs.angular1xExamples
* Mashupjs.angular2xCore
  + When this is complete move the current solution to Mashupjs.angular1xCore and move the 2xCore to the Mashup src
* **Mashupjs.compositeApps**
  + Include gulp tasks that pull projects from other source control directories and imports them.
  + This allows development teams to work independently of each other and combine code only for test/prod deployments.
* Mashupjs.BootstrapExamples
* Mashupjs.ngWidgets
* Mashupjs.wijmo
* Mashupjs.kendoOpenSource
* Mashupjs.[wrapBootstrap-template]
* Mashupjs.Azure
  + Use the mashup to build a UI that demonstrates various Azure design patterns.
  + Cache for reads, queue for writes, SignalR for notifications, Service Bus for Pub/Sub integration
  + Make some nice diagrams, explanations, and follow up with a tutorial on setting it up. (Multi-Part)
  + Create an example app demonstrating it and the differences when the arch is not used. See if there is a demonstrable difference.
* **New multi-session-example**
  + Create a copy from the working core
  + Add app3 – create a template in Yomen or Yo for this.
  + Yoeman – Get templates set up for new app and new page.
  + Add WebApi.AuthADSP to solution
  + Add DAL call the get session from AuthADSP /w AD only
  + Add app 4 with session similar to what we have today combining both AD and local user configuration.
  + Add npm publish for new multi-session-example project
  + Update session and routing documentation
  + *Use the multi-session approach and create a session for each current auth type.*
    - *Link to token based AuthN.*
    - *Link to owin aps identity.*
    - *Link to Identity Server 3.0 AuthN.*
  + *http://bitoftech.net/2014/06/01/token-based-authentication-asp-net-web-api-2-owin-asp-net-identity/*
  + *http://www.codeproject.com/Articles/784106/AngularJS-Token-Authentication-using-ASP-NET-Web-A*
* Mashupjs.signalR
* Mashupjs.hybridMobile
* Mashupjs.ionic
* Mashupjs.advancedRouting
  + Demonstrate advanced routing of RouteUI and/or Angular2’s new router
* Mashupjs.3dui
  + Build a 3d UI using whatever 3D framework is popular
* Mashupjs.Firebase
  + Demonstrate real-time communication with Firebase.
* Mashupjs.EFapi
* Mashupjs.oDataApi

**Yomen**

* Create Yomen tasks
  + Create mashup
  + Create new app
  + Create new page

**Raygun**

* Add Raygun for exception tracking

**Application Instrumentation**

* Consider App Insights from Azure but also look at other products

**Offline First**

* Find a reliable library that uses IndexedDB
* Research AngularFire (Firebase) to see how that fits.

**UI Components Review**

* Flexgrid, AngularGrid, Flexbox, WinJS
* wrapBootstraps package UI components

**Caching Options**

* Angular 1.4 cache options?
* Angular 2.x cache options?
* Resource caching. IE: ajax calls are cached by some library and pools updates that fail when disconnected.
* What popular libraries might help?
* AngularJS 2.0 is coming out with a new persistence layer including “always offline”.
* “Persistence layer: A new persistence layer provides a clean structure for working with servers and local persistent data in the browser, such as “always offline” modes and RESTful service use cases”

**Readme.md**

* Continue updating sections of the Readme.md file that haven’t yet been documented.
* Getting Started: Include notes about the new “dist” folder.
* Update the Readme.md of each repository to give information on how to use the repository and explain the specific purpose of the repository.

**Style Guide**

* Include notes about the new “dist” folder.
* Update on gulp usage in general.

**Blogs**

* Update all blog headers to be consistent, point to my web site and the mashup web site.

**Updating the Mashup Core**

* Document how to update an implementation with Core changes.

**Testing**

* Get a testing solution up and running. Add to the continuous build process/gulp.
* Unit testing
* Automated testing
* Image screenshot testing
* Screenshots for testing responsive design
  + <https://github.com/sindresorhus/pageres>

**Get MashupJS/Readme.md document done**

* Write Workflow document
  + Below can be individual documents but each should be mentioned in the workflow.
  + Creating mashup
  + updating core
  + Publishing/installing to/from npm & other tools like Bower. Need to figure this out.
  + gulp tasks
  + versioning strategy
  + (Plus the list or workflow items in the Readme document now.)
* Create architecture and components document
  + Start with the Visio and other documents already created.
  + Reference or use the Visio doc (mashupcomponents.vsdx)
  + Notes in Mashup-Document “Architecture Notes”
* Create “Using the Mashup for the Enterprise”
  + Notes in Readme.md
* Create - Declarative vs Imperative after code is uploaded
* Features section
  + Create “Lazy loading of HTML templates and JavaScript modules”

**Add to User Manager/Administrator Screens**

* Ability to direct instrumentation and exceptions
* Build dashboard for this data
* Allow each app to have its own config for each user
* There will be a global admin account that can add admin access to other accounts for each app. The admins for each app can add/remove that apps roles/privileges.
* Each app can use any attributes of the session[] object to manage authorization so authorization can be managed by roles in the database, groups in AD, or a combination of both. IE: AD authenticates the user and a table provides the rest of the authorization.

**Menu Tasks**

* Add a sort order/position to the JSON for the menu to use.
* Add items to a different menu bar. Possibly add an attribute that indicates which menu an item goes on so if there are different menus or types of items on a menu the author can redirect the link there.
  + IE: 1 item goes on the dynamic menu, 1 item is the About link on a static menu, 1 item is a stand-alone button next to the menu. Who knows? Just add the attribute for options.
* Add color or some other bootstrap indicator allowing the developer to have some influence over menu items. This might be something that is done by the menu.controller or can be driven by JSON.
* Hide menu items the users doesn’t have access to if their session doesn’t allow it. If their session isn’t yet created then allow the link. App level security with redirect the user to a login page.
* Add Attribute type so the UI knows if it’s dealing with a container or a menu item.
* Figure out how to cache the menu so it’s not built over and over. This only applies if the menu is a popup menu.
  + Might need to only hide popups instead of closing it. That might reduce the extra processing.
* Update the menu so one click event handles all menu items rather than putting an event handler on each menu.

**Log System/Viewer**

* Design/Build solution for collecting logs from client machines
  + Build log viewer
  + Use polling or SignalR to inform the client it needs to send logs
  + Add transmit button to client to send logs as a single batch
  + Add performance data log by intercepting WebApi calls
    - Make into a drop in module like the logService
    - Consider using the heartbeat as a common means to measure performance between applications and to track overall performance.

**Service Workers**

* Implement Service Worker as a replacement for HTML5’s appCache
  + <http://www.html5rocks.com/en/tutorials/service-worker/introduction/>
  + The ability of an application to function off-line or when connectivity is lost.
* IndexDB, project below, needs to be a part of this.
* There are ways to handle this. We need a design pattern to make this implementation consistent, reliable, and easy to code.
* Info from the w3.org
  + HTML5’s [**ApplicationCache**](http://www.w3.org/html/wg/drafts/html/CR/browsers.html#appcache) enables access to Web applications off-line through the definition of a manifest of files that the browser is expected to keep in its cache; while relatively well deployed, the current approach has shown some strong limitations and the HTML and Web Applications Working Groups are considering a potentially major overhaul of the technology, likely based on [ServiceWorker](https://slightlyoff.github.io/ServiceWorker/spec/service_worker/)
  + A [JSON-based manifest format](http://www.w3.org/2008/webapps/manifest/) in development by the [Web Apps Working Group](http://www.w3.org/2008/webapps/). The System Applications Working Group was building a [runtime and security model](http://www.w3.org/2012/sysapps/runtime/) on top of that packaging, but is now instead defining an [application lifecycle specification](http://www.w3.org/2012/sysapps/app-lifecycle/) based on top of Service Workers.

http://blog.kevinchisholm.com/html5-javascript/getting-started-with-html5-web-workers-part-i/

http://blog.kevinchisholm.com/html5-javascript/getting-started-with-html5-web-workers-part-ii/

http://blog.kevinchisholm.com/html5-javascript/getting-started-with-html5-web-workers-part-iii/

**Global exception handling (javascript)**

* Enhance error logging. See if this is something we can/should use.
* <http://www.bennadel.com/blog/2542-logging-client-side-errors-with-angularjs-and-stacktrace-js.htm>

**Communication Protocol (Socket.IO)**

* See if this can replace normal http calls
* What would the pro/cons be for updating our XMLHttpRequest to WebSocket API?
  + The WebSocket API, built on top of the IETF WebSocket protocol, offers a bidirectional, more flexible, and less resource intensive network connectivity than XMLHttpRequest.
* <http://socket.io/blog/introducing-socket-io-1-0/>
* http://socket.io/blog/socket-io-1-1-0/#
* Creator of Socket IO <https://www.youtube.com/watch?v=_8CykecwKhw>
* Microsoft: <http://www.iis.net/learn/get-started/whats-new-in-iis-8/iis-80-websocket-protocol-support>
  + <http://msdn.microsoft.com/en-US/hh673567.aspx>
  + <http://msdn.microsoft.com/en-US/hh969243.aspx>

**Tombstoning**

* The ability of an application to function off-line or when connectivity is lost.
* IndexDB, project below, needs to be a part of this.
* There are ways to handle this. We need a design pattern to make this implementation consistent, reliable, and easy to code.
* Info from the w3.org
* HTML5’s [**ApplicationCache**](#appcache) enables access to Web applications off-line through the definition of a manifest of files that the browser is expected to keep in its cache; while relatively well deployed, the current approach has shown some strong limitations and the HTML and Web Applications Working Groups are considering a potentially major overhaul of the technology, likely based on [ServiceWorker](https://slightlyoff.github.io/ServiceWorker/spec/service_worker/)

**Dirty Page Detection**

* We need an easy to implement method of detecting when data has been changed. When the users tries to leave we should have a way to let the user know their data isn’t saved and give them options to act.

**Web API Global Error Handling**

* <http://www.asp.net/web-api/overview/web-api-routing-and-actions/web-api-global-error-handling>

## DevOps Tasks

* Continuous Build
* Continuous Integration/Deployment
* Continuous Delivery
* Add to build options: <https://scotch.io/tutorials/speed-up-your-deployment-workflow-with-codeship-and-parallelci>

## Possible Blogs

(Items that might become MashupJS documents but might not)

* + Governments willingness to use Cloud
    - <http://azure.microsoft.com/en-us/features/gov/>
  + An app architecture using Azure for resilience and speed.
    - Read from cache
    - Write to queue
    - Somehow work in service bus? Might add complexity and not benefit.
  + Database types (Relational, Key/Value, Column Family, Document, Graph)
  + Blog an ongoing list of job areas emerging. If you can’t learn them all then master something. UI /w SPA, WebApi, Database, Bootstrap, DevOps(continuous build, continuous integration, continuous deploy), Cloud, css (css, less, sass), security, big data mining, IOT, 3DUI, hololens type virtual overlay,
  + When and where to use Filters… many style-guides don’t allow filters
  + IIFE – Don’t pollute the global namespace
  + CSP (Content Security Policy)
    - <https://developer.mozilla.org/en-US/docs/Web/Security/CSP>
  + Responsive-Viewport widths
    - http://i-skool.co.uk/mobile-development/web-design-for-mobiles-and-tablets-viewport-sizes/
  + 3 state check box control
    - Notes in Mashup-Document “Check box”
    - There are angular controls that do this now.
  + Modal popups – add do the angular or bootstrap demo.
  + Browser Link
    - Notes in Mashup-Documentation “Browser Link”
  + $APPLY
    - http://jimhoskins.com/2012/12/17/angularjs-and-apply.html
    - http://angular-tips.com/blog/2013/08/watch-how-the-apply-runs-a-digest/
  + $TEMPLATECACHE
  + STANDARD FORMS
    - Notes in Mashup-Documentation “STANDARD FORMS”

## Build Class