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

# Tasks

## One Off Tasks

* 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

## Projects

* Research AngularFire (Firebase) to see how that fits. Want to make sure cacheService is still relevant.
* Are there other more native caching options I should be using instead?
* Is there a micro-library for offline first design?
* Find a control library(s)
  + Flexgrid, AngularGrid, Flexbox, WinJS
  + Or try out a wrapBoorstrap package

*While working on this keep notes somewhere for design documentation and how to use this.*

Update Getting Started documentation. Everything now runs out of the “dist” directory. This is not obvious and needs documented.

Update all blog headers.

Update the Readme.md of each repository to give information on how to use the repository and explain the specific purpose of the repository.

Document/Blog architecture for highly resilient/performant solution

* 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.

Break out existing example projects.

* WebApi AD project
* Possibly combine these to demonstrate how to combine apps. Could be a cool tutorial.

Update all components

* Document how to update an implementation with Core changes.

Create a couple new skins

* Document how to do this.

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

JS References in VS

* Add and document how to add JavaScript library references in Visual Studio.

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

Start working up example projects to continue my learning

* Study Bootstrap
* Study WebApi
* Study Different database types
* SignalR
* TypeScript
* SASS
* Jasmine for testing

**WebApi project**

* Move/copy the existing routeConfig.js version to the WebApi project that uses the sessionLoad for authentication.
* **Create multi-session system**
  + Update the WebApiAuth example
  + Create a document and link to mashupjs\readme.md
  + Create a property of the session collection object for holding generic, application wide, information.

**Ability to manage environments**

* Add environment to the session allowing each app to choose how to access its own data.
  + Notes in Mashup-Documentation “MASHUP ENVIRONMENTS”
* Update the UI to display the environment

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 **(Depends on WebApiAuth project)**
  + publishing/installing to/from npm & other tools like Bower. Need to figure this out.
  + grunt/gulp tasks **(Depends on grunt rehavilitation)**
  + 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.
* Notes in Mashup-Documentation “ADMIN SCREENS”

Create multiple team workflows (create npm package and readme.md for each)

* Single app in each mashup
* Multiple apps in a single mashup
* Separate teams working with Singles apps that are combined for production
* Hybrid application scenario

Core ability to link to multiple CSS styles as user or admin selection

## DevOps Tasks

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

## Documents to Create

* Create – Gulp/NPM managers (notes in mashupjs\readme.md)
* Create – Wiki
* Create – Learning Path document for the Mashup
* Create document for each, add to philosophies, and update style-guide for:
  + continuous improvement, CI, continuous enhancement, mobile first, offline first
  + continuous deployment, DevOps,
    - Document Offline Patterns  
      <https://www.ibm.com/developerworks/community/blogs/worklight/entry/offline_patterns?lang=en>
    - Offline First Design  
      <http://alistapart.com/article/offline-first>
* Create – Bootstrap document
  + Notes in Mashup-Document “Bootstrap”
* Create – A bunch of Angular tutorials!
* Create - DATA SERVICES
  + Notes in Mashup-Document “DATA SERVICES”

## 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
  + $Log
    - Using and/or overriding
  + Module.run()
  + Web Api Help Pages
    - This will be helpful for other developers and for testing your Web Api.
* <http://www.asp.net/web-api/overview/creating-web-apis/creating-api-help-pages>
  + TODO sometime
  + STANDARD FORMS
    - Notes in Mashup-Documentation “STANDARD FORMS”
  + Markdown
    - Notes in Mashup-Documentation “MARKDOWN”
  + Web Components
    - Created by the Polymer team at Google? (Double check facts)
    - http://www.w3.org/TR/components-intro/#introduction
    - http://webcomponents.org
    - <http://webcomponents.org/resources/>
  + TRANSPILERS
    - Notes in Mashup-Documentation “TRANSPILERS”
  + POLYMER
    - Notes in Mashup-Documentation “POLYMER”

## tasks

* 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.
* Create a help system. Allow the user to enter a question/feedback and for someone in IT to subscribe to the help message.
  + Use SignalR for notifications back and forth. Not really a chat session but a means of providing feedback.
* Deploy app to Azure (require new skills and new documents on Azure)
* Create Mashup Implementations
  + Link to core
  + Link to WebApiAuth example app
  + Link to Multi-Session example app
    - Add user management screen
  + Link to token based AuthN. Link to owin aps identity. Link to Identity Server 3.0 AuthN.
    - STUDY THIS AUTHENTICATION AND MAKE ENHANCEMENTS TO MashupJS
    - <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>
  + Link to Ionic hybrid example app
  + Link to advanced routing example app
  + Link to various teaching example apps
  + Link to 3D UI example apps
  + Link to Angular /w React example app
  + Link to ES6 app
  + Link to ES7 app
  + Link to SignalR app for real-time communication
  + Link to Firebase app for real-time database communication
  + Link to BootStrap example app
  + Link to [examples/libraries/practice]
  + Link to Off-line – behavior of app should be different than a connected app.
  + Like to app with a control library. Possibly a few different control libraries in different apps. Material design.
* Update the shared user alert implementation. Toaster or something.
* 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.
* Background Lazy loading
* Add ng-stats to the core - <https://github.com/kentcdodds/ng-stats>
* Look into ng-inspector
* Implement CSS preprocessor. (SASS)
* Implement TypeScript
  + <http://blorkfish.wordpress.com/2014/02/03/setting-up-typescript-and-angularjs-in-visual-studio-2013/>
* Add testing to core and apps
  + Unit testing
  + Automated testing
  + Image screenshot testing
  + Screenshots for testing responsive design
    - <https://github.com/sindresorhus/pageres>
* Enhance Caching
  + Research all the latest caching options for AngularJS
  + Look at Angular 1.3 to see what options we have.
  + Look at Angular 2.0 to see what’s coming.
  + Look at Angular-Cached-Resources and see what it’s doing in response to Angular 2.0
  + Make a decision and build the new caching for WebApi calls especially for updates.
    - Consider building a queue that feeds whatever system I adopt.
  + If the connection speed is sparatic or slow then create a method or pattern that can be used to compress the package to be transmitted. Maybe this is only for log. The problem is this would have to exist on the webapi side as well so might need to be a custom thing for specific endpoints or a parameter that’s passed to indicate compression. Something to think about. Also consider that compression requires processing power so it’s only worth it if the connection speed is slow.
* 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.
* 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”
* Start building up a code review check list
  + Either add to style guide or as a separate document
* Add names to anonymous function
  + Fn.displayName – might not work in all browsers
  + Add solution to style-guide
* Upgrade to Angular2.0
* Remove YDN dependency
  + Probably best to move to direct IndexedDB but consider this new library from google, Lovefield.
  + <https://github.com/google/lovefield/tree/master/>
  + Possibly replayce YDN with Alasql
    - <https://github.com/agershun/alasql>
    - This has some Angular integration built in and does natural SQL.
* Upgrade the battery indicator with some visual
* Global exception handling
  + 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>
* Add feedback module so users can let us know when things aren’t going well or look wrong.
* Add an indicator for network connectivity
* Create a directive to display local date when given a UTC date.
  + Might already exist I AngularJS
* Implement AngularLint
* Research async design patterns:   
  <http://tech.pro/blog/1402/five-patterns-to-help-you-tame-asynchronous-javascript>
* Review Angular Style Guidelines: <https://github.com/mgechev/angularjs-style-guide>
* Review John Papa’s style guide
* Create Mashup templates with <http://yeoman.io/>
  + Apps, pages, and maybe the core with options.

Pretty URLs:

* + <http://scotch.io/quick-tips/js/angular/pretty-urls-in-angularjs-removing-the-hashtag>

Polymer

* + This is a technology we need to keep our eyes on. In Alpha/Beta now and will be released around Chrome 36.
  + This is a google technology to extend the HTML elements from their base to more complex declarative elements. Even includes data binding with local storage. Very cool potential for reducing the amount of code we must write.
  + http://www.polymer-project.org/
  + https://www.youtube.com/watch?v=8-Zq2KUN6jM
  + https://www.youtube.com/watch?v=p1NpZ-0Op0w

Autocomplete UI in AngularJS

* + This might be the same as “Lazy loading dropdowns” above but this looks like a specific implementation.
  + Investigate this implementation, get it working, and determine if this is our best option for lazy loading/autocomplete.
  + http://www.thumbtack.com/engineering/introducing-smarty/

Communication Protocol (Socket.IO)

* + 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>

Web Worker

Learn how to leverage Web Workers in general and then within AngularJS.

* + Web Workers load up a js file and execute it.
  + Can pass data to the worker thread.
    - Might be able to make a generic thread for data access where we pass only the Web Api string and using that string the Worker updates the cache and the UI.
    - Possibly make a call to get the cache first and then refresh the client and cache after the longer running query quits.
* <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/>

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/)
* 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.
* 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”

Analysis tools

* Learn how to use Analysis tools to check performance of the mashup.
* <https://medium.com/angularjs-collection/3a12a711c7ec>
* Look for memory leaks
* Gather JavaScrip CPU Profile
* Heap Snapshots
* Record Heap Allocations
* DIY: Performance data collection from our users browsers
* <http://cruft.io/posts/rum-waterfall-charts-w3c-resource-timing-api/>
* <http://www.lognormal.com/boomerang/doc/>
* <http://www.sitepoint.com/introduction-resource-timing-api/>
* Timing API
* Resolution Time API
* User Timing API

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.
  + Option 1
    1. On some kind of exit event use interception to stop the process, check for changed data, and stop the process if data has been changed.
    2. Let the user know they have unsaved data and give the option to stay, save and leave, cancel change and leave.

Speech API

* <http://www.sitepoint.com/introducing-web-speech-api/>
* I can see this being useful for grabbing quick notes to reference later.
  + IE: Proactive can have a raw notes field that is populated by whatever a user says after they press the “speech to text” button available when using a mobile device. This information can be used later to complete the Proactive complaint.

Web API Global Error Handling

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

Http Response Message

This should be pretty easy to get going. We just need to take a little time to figure it out, how return the message and what to do with this information on the client if anything at all.

* <http://www.asp.net/web-api/overview/creating-web-apis/creating-api-help-pages>
* http://www.asp.net/web-api/overview/web-api-routing-and-actions/action-results
* <http://www.asp.net/web-api/overview/working-with-http>

HttpGet, HttpPut, HttpPost, HttpDelete

* Some time was spent researching a reason to choose which of these types of http requests to make. So far the best answer is for supporting multiple types of calls on a single subject.
  + IE: If a subject is “car” then a simple URL path accompanied by could indicate intention.
* With Attribute based routing these options become less valid and add a level of complexity. Not only does the path show intent but its intent is further modified by the type of Http call.
* Based on the options for passing data, for now, we will perform http request as Post
  + This is only a default position. The other http request types can be used at any time.
  + HttpGet can use values passed as part of the URL path.
  + HttpPut, HttpPost, HttpDelete can use values passed in the URL path and Body.
  + With HttpPost, for example, we can perform all crud operations and we have access to all values passed FromURL and FromBody.
* UPDATE
  + “Results of GET operations can be cached, which means that multiple calls to the same operation may result in only one request to your service.”
  + Based on this information we should use Get for all fetches where simple types are used. If a very complex type is required then we can drop back to Post/Put to pass the data in the body.
  + HttpGet – all simple selects because it can be cached.
  + HttpPost – all queries that modify data.

## Build Class