---

title: Mashup/Apps

tags:

- AngularJS

- Bootstrap

- Router

- Menu

- AuthN

- Auth

- Authentication

- AuthR

- Authorization

- HTML Template

- Resolve

- sessionService

- cacheService

---

Mashup/Apps

One of the functions of the Mashup is allowing you to build new applications without having to build the plumbing again. Core features and libraries are available to all Mashup applications.

The apps directory is for your applications. After installing the Mashup you’ll have a couple starter apps as a map for how to create your own.

Placing your application in the Apps folder and using a few conventions makes integrating apps into the Mashup seamless.

# Routing

route.config.js

Route configurations placed in this file are combined by Grunt/Gulp and loaded at run-time.

This approach makes it possible to drop in or remove applications without having to fiddle with routing. Each app is 100% self-contained and can be moved easily between MashupJS implementations.

# Menu

menu.config.js

Describes the intended menu structure in JSON. The menu structure can be static or you can create a process that dynamically generates it based on user roles/rights.

This file is concatenated with other menu.config.js files and load at run-time.

# AuthN/AuthR

An example of a basic authentication method is implemented in the [root]/apps/mashup.

Each application is responsible for its own security but each application can subscribe to the user session of another application. It’s likely, in companies using AD, only one session is created from which app applications derive authentication and authorization properties.

Authentication and authorization are performed in the route configuration using “resolve”.

Example:

mashupApp.config(['$routeProvider', function ($routeProvider) {

$routeProvider.otherwise({ redirectTo: '/mashup' });

$routeProvider

.when('/mashup/about', {

templateUrl: 'apps/mashup/about.html',

controller: 'mashup.AboutController',

controllerAs: 'vm',

**resolve: {**

**loadMyCtrl: ['$ocLazyLoad', function ($ocLazyLoad) {**

**// you can lazy load files for an existing module**

**return $ocLazyLoad.load({**

**name: 'mashupApp',**

**files: ['apps/mashup/about.controller.min.js']**

**});**

**}],**

**resolveRoute: ['$route', 'mashupRouterAuth', function ($route, mashupRouterAuth) {**

**return mashupRouterAuth.resolveRoute(['Administrator']);**

**}],**

}

})

The “resolveRoute:” function is executed before the route can be resolved. If the user is not authenticated then they can be re-routed to a login page. If the user is not authorized then they can be routed to a page that says they are not authorized. The resolveRoute function is injected with the “mashupRouterAuth” which gives access to the “resolveRoute” function. The “**mashup**” in “**mashup**RouteAuth” is referring to the name of the app plush “RouterAuth”. You application, if named “accounting”, could be “**accounting**RouteAuth”.

You can deviate and improve upon this basic design.

# Sessions

Each application can have its own session or share. It’s possible that all your applications use one session except for a customer facing application that uses Identity Server 3. The mashup can easily accommodate multiple sessions.

There are two different types of session. There is the “sessionService” and an application’s user session.

The sessionService is for general use by utilities such as the logService. Only a little user information is maintained in the sessionService to let utilities know the user and application that was being used at that moment. When switching to another application within the mashup the user id from that session and its application name are updated within the sessionService.

The application’s user session is stored in IndexedDB and retrieved by the session name.

**Basic AuthN/AuthR Example:**

(function () {

**getAppSession**().then(function (data) {

var appUserSession = data[0];

var session = \_.first(\_.where(appUserSession.sessions, { 'appName': **'coreSession'** }));

var isAuthenticated = isUserAuthenticated(session);

var isAuthorized = isUserAuthorized(session, authGroupArray);

if (!isAuthorized) {

// Just kill the page change completely.

defer.reject();

}

if (!isAuthenticated) {

// HERE YOU CAN SET $location.path('/login') to force authentication.

$location.path('/mashup/login');

}

else {

session.sessionLastUsed = utility.localMilToUtcMil(new Date().getTime());

}

coreRouteHelper.logRoute('mashup');

defer.resolve(true);

});

})();

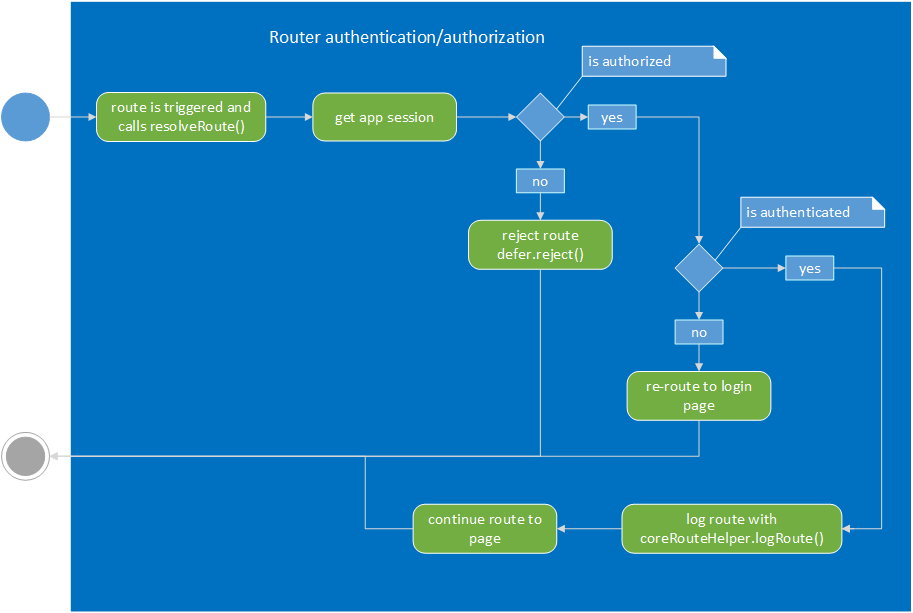
return defer.promise;

};

var **getAppSession** = function () {

return cacheService.getCache('mashupSessions');

};



# JavaScript Loading Options

Depending on your development and delivery workflow there are multiple approaches to optimizing and building your solution.

* Use Grunt/Gulp to create one file for each page Lazy load each JS as needed.
* Use Grunt/Gulp to create one file per application.
* Use Grunt/Gulp to create one file for entire mashup and apps.
* Initial load time versus deep linking.
  + You can optimize the initial load of the first page but when users can deep link into any place in the application the quick initial load is lost. The option I’ve chose is lazy loading components needed for any page via the router. This gives us a fairly quick deep linking load time.
  + These are line-of-business applications that are used repeatedly. Once the application has loaded once the follow up loads should pull scripts from cache.

# apps/mashup

Welcome Page – just a landing page.

## About Page

Displays all sessions with one tab per session and the session in the form of JSON. You’re about page will have a more custom display of user session information.

The About page also displays what is currently cached for fast data retrieval.

## Login Page

The login page demonstrates a basic way to get and store authentication information as a user session.

