MONAHRQ Generated Websites

The MONAHRQ website is generated through the desktop application by the Host User. It provides a healthcare reporting website that dynamically customizes itself based on the data and configuration provided by the Host User. It is split into two sub-sites.

The *Consumer* site is the portion of the website tailored for consumers of health care services, with simplified and relevant messaging, reports, and UI affordances. The *Professional* site is targeted towards health care professionals, with more technical and detailed information suited to their needs.

The website is a single page application built using the AngularJS v1.4 JavaScript framework. It relies heavily on the capabilities provided by the Angular ecosystem, such as controllers, services, directives, and routing. Key JavaScript and CSS libraries which the developer should be familiar with include AngularJS, ui-router, underscore, bootstrap, and SCSS.

The web application's toolchain and build processes are separate and distinct from those of the host application. The web app is built with a Node.JS toolchain and deployed to a web server. The host application is a .NET application that installs to the Host User's workstation, and is used to generate the data (Wings) that the web application reports on.

Development Copy

The development copy of the website template is not distributed with MONAHRQ. It may be found in the source code under `MONAHRQ\Resources\Templates\Site.Src` and includes build scripts and additional development dependencies.

Development Workflow

To begin development of the web application, it is first necessary to install the desktop application and generate a website through it. This will create the data files from your Wings. These data files are then available for use by the in-development version of the web app, and further site generations are not required to customize it.

The workflow for customizing the web app will be familiar to those with experience building JavaScript single page apps. A typical change would be developed as follows:

1. Obtain desired version of MONAHRQ
2. Copy a `data` directory from a previously generated site into `Site.src/src`
3. Make desired changes in `Site.src/src`
4. Run `grunt build` to update the distribution template used by MONAHRQ

Website Template for Distribution

The website template distributed with MONAHRQ may be found in the `MONAHRQ\Resources\Templates\Site` directory. This template contains optimized templates and

The website template does not contain any data and is therefore not functional on its own. To work directly with the template, copy a `data` directory from a working MONAHRQ website into the template's `src` directory.

dependencies and is copied as-is to generated websites.

External Runtime Dependencies

Dependency	Purpose
Google Analytics	Tracks End User behavior on the site
Google Maps	All map visualizations, such as those in the Map tab of the hospital search results page, are provided by Google Maps
Physician Compare	Medicare API that provides information about physicians in the United States; used to power consumer and professional physician reports
MapQuest Geocoder	Used by the consumer website to translate End User-provided locations into latitude and longitude coordinates for searching

Architecture

MONAHRQ's generated websites are segmented into several key modules, which are described below.

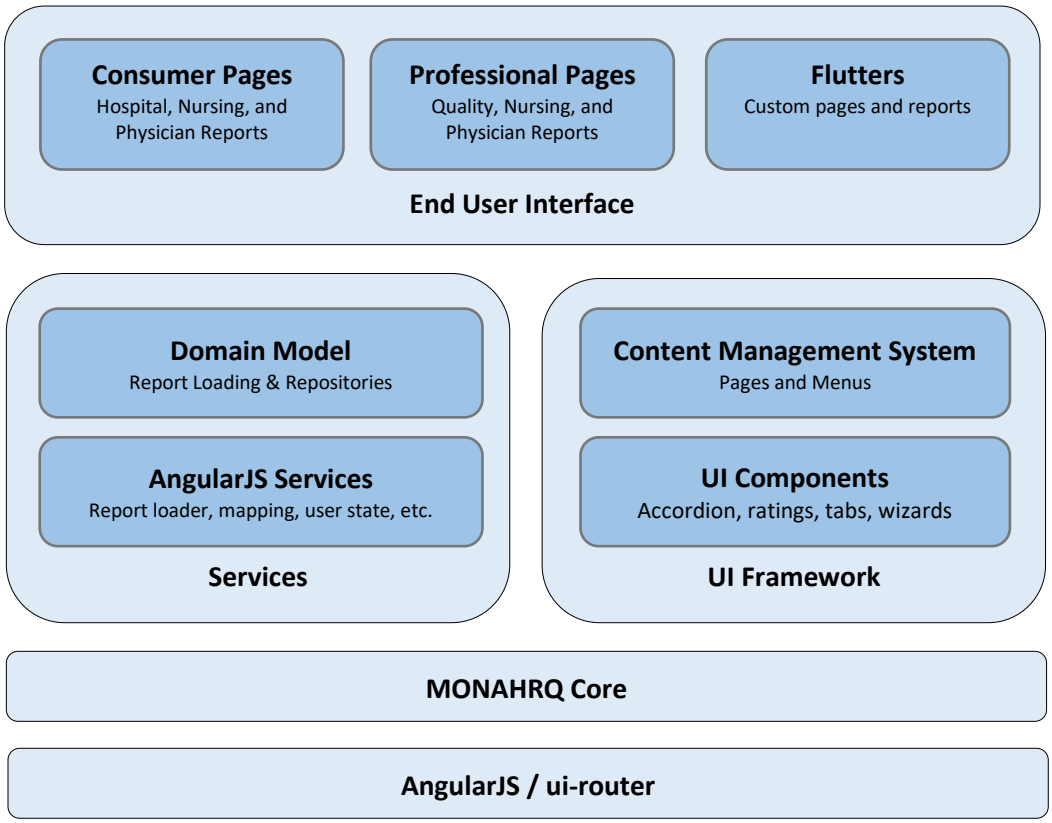

Figure 1 Generated website architecture

Core

The *core* module has a small footprint, and is focused on bootstrapping the application. It handles tasks such as loading 3rd party and MONAHRQ modules, initializing Flutter plugins, loading essential configuration and data, global logging and error handling, managing top-level classes for styling, etc.

Services

The *services* module provides a variety of Angular services whose functionality is intended to be used across the application. This includes tasks such as low-level data loading, string formatting, geo math functions, and sorting.

UI Components

The *UI Components* module consists of a diverse array of Angular directives offering reusable UI widgets and behaviors. Some widgets are generic, such as an accordion, autocomplete dropdown, menus, modals, and tabs. Others are focused on domain-specific use cases, including charting, content management, user help, and quality rating iconography.

Domain Model

The *Domain Model* module provides services for working with application data. This includes loading, processing, and querying Wing reports, and loading Base data files.

CMS

The *CMS* module provides an avenue for Host Users to customize the textual content of the website. They can modify the navigation menus, edit the content of static pages such as “about” and “help”, and insert headers and footers onto report pages.

Flutters

The *Flutters* module is a means to create custom plugins to extend the functionality of the website. Developers can create completely custom reports using new or existing Wings as a data source, and hook access to these reports into the website navigation.

Consumer and Professional

The *Consumer* and *Professional* modules implement their respective sections of the website. They share many of the same underlying Wings, components, and data services, but their Angular controllers and views are entirely separate to provide the best experience to each audience.

Directory Structure

Directory	Purpose
app	AngularJS application code
components	Custom AngularJS directives
core	Application bootstrapper that loads modules, key data files, etc.
domain	The domain model of the application: from lower-level data loading services to repositories for base, hospital, nursing home, and physician report data
lib	Third-party libraries not managed by a package manager
products	AngularJS states, controllers, and views used to produce each of the pages available on the web site
consumer	Sub-site intended for consumer use
components	Custom AngularJS directives specific to the consumer sub-site
hospitals	Reports
nursing-homes	Reports
pages	Non-report pages and reusable templates
physicians	Reports
professional	Sub-site intended for use by professionals
nursing-homes	Reports
pages	Non-report pages and reusable templates
physicians	Reports

quality-ratings	Reports
usage-data	Reports
services	Utility functions used by the application, including sorting collections, string formatting, and geographic calculations
vendor	Third-party libraries managed by a package manager
data	Base (lookup) data, and reports generated from Wings
flutters	Reports generated from installed Flutters
theme	CSS styling
base	Shared CSS stylesheets
consumer	CSS stylesheets for the consumer sub-site
professional	CSS stylesheets for the professional sub-site

Section 508 Compliance

The majority of the code required for section 508 compliance may be found in the components under `app\components`.

CSS-related accessibility fixes may be found under `theme/consumer/_508.scss` and `theme/professional/_508.scss`.

Cross-Browser Compatibility

The file `theme/consumer/_ie.scss` maintains compatibility tweaks required for Internet Explorer versions 11 and earlier.

Request Routing

The web application uses the UI-Router (https://ui-router.github.io/) library to translate End User interactions to the corresponding application pages and states. The core application manages several top-level states, while individual modules are responsible for configuring the states specific to their scope.

The root state of the application is *top*, defined in `core/app.js`. All other states are descendants of this state. Its purpose is to load core data files. The file, `websiteConfig` is the core configuration, specifying details such as global behavior, which products are available and their properties, API keys, etc. `reportConfig` specifies the properties and behavior of individual reports. `menu` details the End User-facing navigational elements on the pages. The bootstrap also loads the Content Management System pages and elements, initializes any Flutter plugins, and starts Google Analytics.

Each product, consumer and professional, then defines a state that all its modules descend from, which is defined in `pages/index.js`. For example consumer defines the `top.consumer` state. This state also loads several UI components shared across the entire product – the page header, footer, and navigation.

The modules within a product, such as `hospitals` or `nursing-homes` within consumer, then define a shared state from which individual reports and pages derive from. This is simply an organizational state – it doesn't use any controllers or data loading.

The individual pages within a module are all straightforward. They define a URL with its parameters, a template, a controller, and any data they need upfront before the controller runs. Additionally, two parameters are specified for each page: `pageTitle` is the title that should be shown in the browser title bar, and `report` is the report id(s) for any reports that live on the page. It is used to retrieve the `ReportConfig` record for those reports.

Loading Data

MONAHRQ's design requirements dictate that generated websites must be able to be loaded from a web server and from the local filesystem. While an application would typically load data via XHR requests, this is not possible in the local scenario due to the same-origin policy imposed by the browser security model.

The solution used by the application is to inject `<script>` tags into the browser DOM, with those tags referencing syntactically-valid JavaScript files containing a particular unit of data to be loaded. The files contain an array or object assigned to a namespaced location within the global window object. All modern browsers support this method of loading data without running afoul of security restrictions.

Data Loader Service

The data loading code is implemented by the `DataLoaderSvc`, defined in `services/data_loader.js`, which provides a standardized method for accomplishing the load, using a single API function:

```
function loadScript(url, callback, errorcallback, forceRefresh)
```

If `forceRefresh` is false, the service will not reinject the `<script>` tag for a URL that was previously successfully loaded.

Report Loader Service

Most of the report loading code does not interact directly with the `DataLoaderSvc`. Instead, report loaders use a simplified interfaced provided by `SimpleReportLoaderSvc`, which allows reports to be loaded in a configuration-driven manner.

SimpleReportLoaderSvc API

It supports data file layouts where a single directory contains a set of one or more data files, which is typical of Wings. The service provides two API functions:

```
function load(configuration, id)
function bulkLoad(configuration, ids)
```

The `load` function is used to load a single data file from the report directory. The id parameter specifies which file to load, in cases where there are multiple data files. The function returns a promise that provides the loaded data when resolved.

The `bulkLoad` function is similar to the load function, except that it accepts a list of ids that all resolve in a single promise.

Both functions accept the same configuration object; the following is a typical example:

```
{
  rootObj: $.monahrq.NursingHomes.Report,
  reportName: 'Measures',
  reportDir: 'Data/NursingHomes/Measures/',
  filePrefix: 'Measure_'
}
```

Figure 2 Typical example of the "configuration" parameter value

rootObj is the where the loaded report is found in the MONAHRQ global data object. `rootObj` is either a JavaScript object reference to the report data, or a string value that is appended to the `$.monahrq.` object.

reportName is the leaf attribute following the object path specified by reportObj. It is expected to refer to either a JavaScript object or list containing the data.

reportDir is the filesystem path to the Wing report data being loaded.

For example of how these properties work, consider a flutter specifying a `rootObj` of `flutters.hcupcountyhospitalstaysdata.report` and a `reportName` of `summary`. The report generator must output the data to a `.js` file in the `reportDir` directory using a JavaScript namespace of `$.monahrq.flutters.hcupcountyhospitalstaysdata.report.summary`.

filePrefix is report filename without the id value and extension. For example, a filePrefix of "`Hospital_`" and an ID number of `25` would yield the filename, "`Hospital_25.js`".

When the promise returned by the `load` function resolves, it returns an object containing the id parameter and report data: `{ id: 25, data: { … } }`. If the report file failed to load — e.g., a 404 — the promise will still resolve, but the data attribute will be null.

The `bulkLoad` function will return an array of the above objects.

Consuming SimpleReportLoaderSvc

The consumer and professional sites share a common domain model, found in `src/app/domain`. Each category of data (hospitals, nursing homes, and physicians) will have at least two services to assist with report loading: `*ReportLoaderSvc` and `*RepositorySvc`.

The report loader service is a thin wrapper around `SimpleReportLoaderSvc`. It provides the correct configuration needed to load the standard Wings generated by MONAHRQ, along with functions for loading specific reports. For example, hospitals have a method named `getQualityByHospitalReports` which will load the quality reports for an array of hospital IDs provided by the caller.

The repository service is used to encapsulate search algorithms commonly used by various pages on the website. For example, hospital has methods for finding reports by the hospital name, id, or within a certain distance of an arbitrary street address. There are other repositories provided as well: both physicians and nursing homes have specialized repositories for working with CAHPS reports, and

physicians has an additional repository for loading data from a remote Medicare-provided web service API.

End User Interface Theming

The `themes/base` directory contains the `bootstrap.css` and `bootstrap-theme.css` files. These files should not need to be modified unless updating bootstrap.

The `themes/consumer` and `themes/professional` directories of the compiled website template contain two key files:

- `consumer.css` or `professional.css`: a compilation of all lower level `.scss` files from the development website
- `user-settings*.css`: a standalone CSS file that may be customized as needed to tweak website colors

The source `.scss` files that make up these `.css` files are located in the development website. In general, one `.scss` file corresponds to one UI component. Notable exceptions to this are the `_508.scss` and `_ie.scss` files used for section 508 compliance and Internet Explorer compatibility, respectively.

To compile the SCSS files into CSS, run `grunt build` in the `Site.Src` folder.

If you'd like to have Grunt monitor the source directory for changes as you make them, use `grunt watch` instead.

Differences between Consumer and Professional Themes

The core of the consumer website theme is located in the `consumer\kentucky` directory. The professional theme does not contain this directory; instead, the `_base`, `_button` and `_mixins` files provide the core styles.

The consumer theme contains four built-in color schemes (`user-settings*.css` files); the professional site contains only one.

User Settings Files

The End User settings files for the consumer and professional themes are generated by MONAHRQ based on the theme and/or colors selected by the Host User. The following area of the site may be customized by the End User settings file:

- Text link colors
- Button background colors
- Main navigation active link color
- Footer background color
- Search hero background color
- Report result background color
- "About the Ratings" background colors

All other colors are hard-coded and cannot be changed by the Host User. Text colors are chosen automatically by MONAHRQ to ensure readability.