



State Farm Litigation

WEX Solutions Overview

Authors:

Vipin Pingle

Apoorv Rajput

June 28, 2017

Contents

[Current State Farm Litigation Solution Overview 4](#_Toc494372776)

[WEX Components Interaction Overview 5](#_Toc494372777)

[Database Views 7](#_Toc494372778)

[Purpose 7](#_Toc494372779)

[Other details 7](#_Toc494372780)

[WEX Engine Search Collections 7](#_Toc494372781)

[AppBuilder Entities 8](#_Toc494372782)

[Analysis 8](#_Toc494372783)

[Invoice 8](#_Toc494372784)

[InvoiceAnomaly 8](#_Toc494372785)

[InvoiceLineItemAnomaly 8](#_Toc494372786)

[Matter 8](#_Toc494372787)

[Firm 8](#_Toc494372788)

[TimeKeeper 8](#_Toc494372789)

[AppBuilder Endpoints 8](#_Toc494372790)

[AnalysisDataEndpoint 9](#_Toc494372791)

[Purpose 9](#_Toc494372792)

[Input 9](#_Toc494372793)

[Output 9](#_Toc494372794)

[AnalysisCharts 9](#_Toc494372795)

[Purpose 9](#_Toc494372796)

[Input 9](#_Toc494372797)

[Output 10](#_Toc494372798)

[AnalysisMatterDropDown 10](#_Toc494372799)

[Purpose 10](#_Toc494372800)

[Input 10](#_Toc494372801)

[Output 10](#_Toc494372802)

[DashboardEndpoint 10](#_Toc494372803)

[Purpose 10](#_Toc494372804)

[Input 10](#_Toc494372805)

[Output 10](#_Toc494372806)

[InvoiceList 10](#_Toc494372807)

[Purpose 10](#_Toc494372808)

[Input 10](#_Toc494372809)

[Output 11](#_Toc494372810)

[InvoiceLineItemsList 11](#_Toc494372811)

[Purpose 11](#_Toc494372812)

[Input 11](#_Toc494372813)

[Output 11](#_Toc494372814)

[User Interface 11](#_Toc494372815)

[Bootstrap Cascading Styles 11](#_Toc494372816)

[JavaScript Libraries 12](#_Toc494372817)

[D3 JS 12](#_Toc494372818)

[Highcharts JS 12](#_Toc494372819)

[Angular JS 12](#_Toc494372820)

[Application UI Architecture 12](#_Toc494372821)

[UI Layouts 14](#_Toc494372822)

[Application Components 15](#_Toc494372823)

[Navigation Widgets 15](#_Toc494372824)

[Dashboard/Home View 15](#_Toc494372825)

[Invoices View 16](#_Toc494372826)

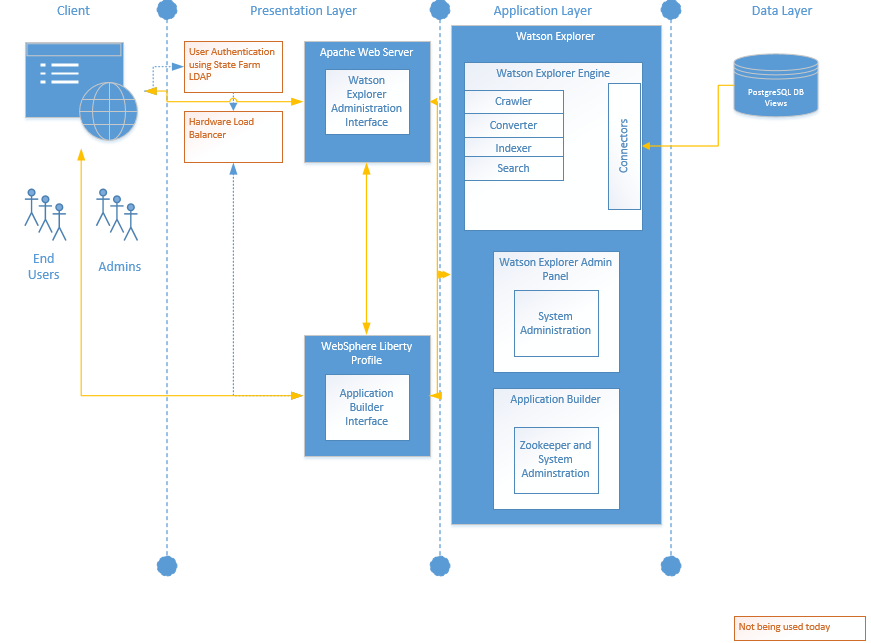
[Firms View 17](#_Toc494372827)

[Matters View 18](#_Toc494372828)

[Team View 18](#_Toc494372829)

[Analysis View 18](#_Toc494372830)

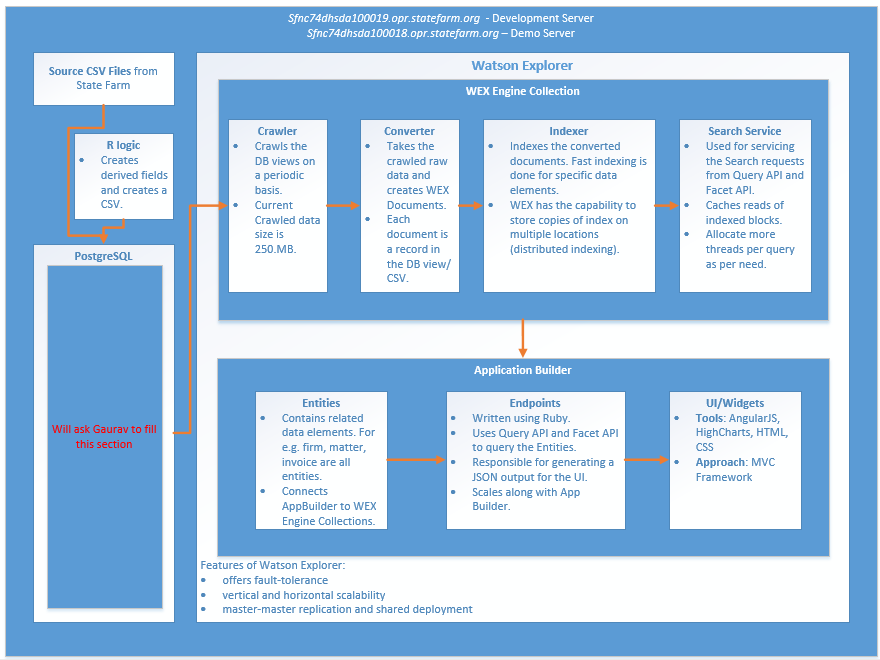
# Current State Farm Litigation Solution Overview



**Connectors:** Connectors use seeds to retrieve data from a data source. Seeds are the programmatic mechanism by which Watson Explorer Engine enables a connector to crawl a resource. All installed connectors in Watson Explorer Engine have a corresponding seed that must be added to the search collection first to enable that connector to crawl the search collection.

**Zookeeper:** A ZooKeeper server or group of servers contains the configuration information (application and entity model) that is common across all Application Builder installations in the deployment. Each Application Builder instance is configured to connect to ZooKeeper and automatically obtains the current configuration information when an application page is requested by the user.

**Watson Explore Administration Interface:** The Administration Interfaces provides a graphical interface that enables configuration of the Watson Explorer Engine software without having to modify the XML files that the Watson Explorer Engine uses internally. The Watson Explorer Engine administration tool also allows viewing the WEX documentation and viewing debugging information (logs).

WEX Components Interaction Overview 

**Fast Index:** Fast-indexing is the general term for indexing methods that enable content to be accessed quickly, and which is flexible enough to be used to filter and sort query results. The Fast Index uses content elements for Ranking, Sorting, and Filtering and Refinements and Binning. The indexer can treat some content elements specially. This is done by fast-indexing those content elements.

**Caching:** The Watson Explorer Engine search engine supports two different types of caching for each search collection to improve and optimize Watson Explorer Engine platform application performance and responsiveness:

* Index data: maximizes the amount of search collection index data that is cached in memory and reduces the amount of index data that must be read from disk
* Previous search results: caching the results from previous searches may reduce the number of times to look for matching results in an index

**Entity:** An entity type contains related data elements such as: firm, matter, employee, time keeper, invoice, or other groups of logically related dated.

**Query API:** The Application Builder Query API is a Ruby domain-specific language used to search and navigate the Application Builder entity model that endpoints access to populate data in widgets.

**Facet API:** The Faceting API is a subset of the Application Builder Query API that groups data in the result set based on metadata.

**Endpoints:** Endpoints are custom functions that access entities and external web services to deliver data for use by widgets and external systems.

# Database Views

## Purpose

The purpose of the DB views is to provide simplified data sources to WEX Collection crawlers. The views are built on top of Facts and Dimensions tables. These tables have been loaded with data from CSV files. PostgreSQL is being used for Database implementation, but the intent is to migrate to SQL Server.

## Other details

Database views are defined in detail in the document titled *Database approach for WEX initial phase.* The database views are crawled by WEX Engine Search Collection Crawlers. The raw crawled data is converted and then indexed. Indices are stored in WEX and are made available to the Search API.

# WEX Engine Search Collections

A search collection is the term used to describe the Database Views source and the online index created from them when the Watson Explorer Engine search engine crawls the Database views. A Watson Explorer Engine search collection is defined using the Watson Explorer Engine administration tool and is crawled and indexed when it is created (and, subsequently, at scheduled intervals). The index is consulted whenever that search collection is queried.

A WEX Engine Search Collection is created for each Database view. Each collection has its own seed that points to a DB view, custom converter to add more metadata to the crawled information, and a set of Fast Indexes configured in the indexing section. Fast Indexing helps with sorting, ranking, binning, and aggregation of data.

**List of collections created:**

* V2\_ANALYSIS\_VIEW: All the drop down UI elements on the Analysis page are sourced from this collection.
* V2\_ANALYSIS\_METRICS\_VIEW: This collection contains the most granular data. It has information about each invoice line item and metadata associated to each invoice line item. The charts are sourced from this collection.
* V2\_ANALYSIS\_ANOMALIES\_METRICS\_VIEW: This collection provides indexed content related to the Invoice Line Item Anomalies. The data from this collection comes into play when the Anomalies KPI is selected.
* V2\_ANALYSIS\_INVOICE\_ANOMALIES\_METRICS\_VIEW: This collection provides indexed content for Invoice Level Anomalies. The content from this collection is used in conjunction with content from V2\_ANALYSIS\_ANOMALIES\_METRICS\_VIEW to display Anomaly specific charts.
* V2\_AVERAGES: This collection contains the mean and median averages for analysis view.
* V2\_INVOICES\_LIST\_VIEW: This collection contains the list of Invoices.
* V2\_INVOICE\_ANOMALIES\_VIEW: This collection contains the list of invoice anomalies.
* V2\_INVOICE\_ANOMALY\_FIRM\_MATTER\_VIEW: This collection contains the invoice anomalies related to firm and matter.
* V2\_INVOICE\_DETAIL\_VIEW: This collection contains the invoice details for each invoice.
* V2\_INVOICE\_DETAILS\_WITH\_ANOMALIES: This collection contains the anomalies for invoice details.
* V2\_INVOICE\_LINEITEM\_ANOMALIES\_VIEW: This collection contains the invoice line item anomalies.
* V2\_FIRM\_AGGRERGATED\_VIEW: This collection contains pre-aggregated information at the firm level. A firm’s detail view is sourced from this collection.
* V2\_FIRM\_TIMEKEEPER\_VIEW: This collection contains pre-aggregated information at the Firm-Timekeeper level. A firm’s Time Keeper view is sourced from this collection.
* V2\_FIRM\_LIST\_VIEW: This collection contains the list of firms.
* V2\_FIRM\_MATTER\_COUNT\_VIEW: This collection contains the list of firms with matter counts.
* V2\_FIRM\_MATTER\_TIMEKEEPER\_TRACKER\_VIEW: This collection contains the list of firm, matters related to timekeepers.
* V\_MATTER\_FIRM\_AGGREGATED\_VIEW: This collection contains the aggregated data for matters and firms.
* V2\_LINE\_ANOMALY\_FIRM\_MATTER\_TIMEKEEPER\_VIEW: This collection contains the list of line item anomalies for firm, matter related to timekeeper.
* V2\_MATTER\_AGGREGATED\_VIEW: This collection contains pre-aggregated information at the matter level. A matter’s detail view is sourced from this collection.
* V2\_MATTER\_TIMEKEEPER\_VIEW: This collection contains pre-aggregated information at the Matter-Timekeeper level. A matter’s Time Keeper view is sourced from this collection.
* V2\_MATTER\_FIRM\_BUDGET\_VIEW: This collection contains the budget data for matters and firms.
* V2\_MATTER\_LIST\_VIEW: This collection contains the list of matters.
* V2\_MATTER\_PHASE\_AGGREGATED\_VIEW: This collection contains the aggregated data for matters related to phase.
* V2\_MATTER\_PHASE\_BUDGET\_VIEW: This collection contains the budget data related to matters and phase.
* V2\_MATTER\_PHASE\_DORMANCY\_VIEW: This collection contains the dormancy data related to matters and phase.
* V2\_MATTER\_YEARLY\_BUDGET: This collection contains the yearly budget data for matters.

*NOTE: A collection is created for each database view*

# AppBuilder Entities

The core of an Application Builder application are the entities that identify and connect the unique information in your business. An entity type identifies a concept such as an invoice, firm, matter, or anything else that can be identified within the business.

A unique substantiation of an entity type is known as an entity instance. For example, John Smith could be an entity instance of the user entity type.

Each entity equates to a single result delivered into Application Builder from one of the data sources. A result contains information in fields describing the entity instance. Each entity created in the Litigation Billings Insights solution maps to a collection in the WEX Engine.

## Analysis

This entity holds the most granular bit of information present in the dataset. The Analysis screen interfaces primarily with this entity to aggregate on the fly, to source filtered drop down values, and to display charts. Each object in this entity is a unique Invoice Line Item.

## Invoice

This entity holds the invoice specific information. Each object in this entity is a unique invoice.

## InvoiceAnomaly

This entity holds the Anomaly details for each Invoice.

## InvoiceLineItemAnomaly

This entity holds the Anomaly details for each Invoice Line Item.

## Matter

This entity holds Matter specific information. Each object in this entity is a unique matter. The Invoice Line Items are aggregated up to the matter level in the DB view to get this information.

## Firm

This entity holds aggregated information for the firms in the data set. Each object in this entity is a unique firm.

## TimeKeeper

This entity holds aggregated information for the timekeepers in the data set. Each object in this entity is a unique timekeeper.

# AppBuilder Endpoints

Endpoints are custom functions that produce data for use by widgets and external systems. Endpoints can access entity data and external web services.

Creating endpoints in the IBM WEX Application Builder administration tool can accomplish the following goals:

* Reuse code for custom widgets. Widgets that need to share data or calculations can use the same endpoint.
* Integrate external services, such as services from the IBM Watson™ Developer Cloud and AlchemyAPI®, with Application Builder.
* Refresh widgets asynchronously by invoking endpoints, and then rendering the resulting data via JavaScript (Ajax).
* Expose Application Builder entity data to external systems.

Endpoints have the following characteristics:

* An endpoint can be accessed internally by the application or externally from a URL call.
* Endpoints are stateless blocks of code that can be accessed from any part of the application.
* All calls to endpoints must be authenticated. Endpoints have access to the current\_user method.

## AnalysisDataEndpoint

### Purpose

AnalysisDataEndpoint reads from the Analysis Entity. This endpoint is called using an AJAX call from the JavaScript code modules. The primary purpose of the Endpoint is to return a unique list of data elements as per request (e.g. list of Firm Names or list of Matter Names etc.). The returned data elements act as a source for the drop down components that exist on the Analysis page.

### Input

This endpoint has 2 input parameters:

1. FilterValue:
2. FindFieldName

### Output

The output of this Endpoint is a JSON array that contains a names of the content element being requested for (in the FindFieldName parameter).

## AnalysisCharts

### Purpose

The primary purpose of this Endpoint is to return a JSON that can be consumed by the HighCharts JS to display Pie Chart, or Bar Chart, or Time Line View, or Time Comparison Chart.

### Input

1. Category
2. Categoryvalue
3. Charttype
4. Datefilter
5. Filters
6. Selectedkpi
7. Subcategory

### Output

The output of this Endpoint is a JSON that is consumable by the HighCharts JS. The format of the JSON differs based on the Charttype parameter being passed.

## AnalysisMatterDropDown

### Purpose

The Matter Name dropdown on the Analysis page is different from all other dropdowns. As per the business requirement, the Matter Name dropdown needs to display the Matter Name and the Matter Number. For that reason, a separate Endpoint has been created to cater the unique requirement for Matter Name dropdown. This Endpoint returns a JSON map which has the key as the Matter Name and the value as the Matter Number.

### Input

1. FilterValue

### Output

The output of this Endpoint is a JSON, which is a list of ‘Matter Number – Matter Name’ combinations.

## DashboardEndpoint

### Purpose

This endpoint serves the Dashboard page. All the charts on the Dashboard page are sourced from this endpoint. This endpoint functions the same way as the Analysis endpoints and aggregates information based on the parameter values being passed to it.

### Input

1. Category
2. Categoryvalue
3. Datefilter
4. Filters
5. Selectedkpi
6. subcategory

### Output

The output of this Endpoint is a JSON, which is a list of subcategory being requested for along with the aggregated KPI value.

## InvoiceList

### Purpose

This endpoint queries the Invoice entity to fetch a certain number of records being requested for. It is responsible for sourcing the Invoice List UI component. This endpoint also takes care of sorting, filtering and paginating the Invoices.

### Input

1. noofresults
2. pagenumber
3. searchstring
4. sortbyfield

### Output

The output of this Endpoint is a JSON, which is a list of Invoices and metadata specific to each invoice.

## InvoiceLineItemsList

### Purpose

This endpoint queries the Analysis entity to return Invoice Line Items specific to each invoice. This endpoint also takes care of sorting, filtering and paginating the Invoice Line Items.

### Input

1. invoiceid
2. matternumber
3. noofresults
4. pagenumber
5. searchstring
6. sortbyfield

### Output

The output of this Endpoint is a JSON, which is a list of Invoice Line Items and metadata specific to each invoice line item.

# User Interface

Application builder is organized into multiple components that are used together to deliver an application to the user. This application is based on the Ruby, JavaScript, HTML, and CSS. A customized user interface is built using following components.

## Bootstrap Cascading Styles

Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web. Bootstrap makes front-end web development faster and easier. It also provides extensive documentation for common HTML elements, dozens of custom HTML and CSS components. The css files for this application are located on server in folder **/opt/ibm/WEX/AppBuilder/wlp/usr/servers/AppBuilder/apps/AppBuilder/dashapp/src/stylesheets**. Below listed files are required for bootstrap to function properly.

* Bootstrap-theme.css - A customized version of the bootstrap CSS file used in this application.
* Bootstrap-theme.css – A customized theme for the bootstrap CSS.
* datepicker.css – A customized CSS for the date picker calendar.

## JavaScript Libraries

The customized application built on the below libraries. The JS libraries are located on the server in folder **/opt/ibm/WEX/AppBuilder/wlp/usr/servers/AppBuilder/apps/AppBuilder/dashapp/src/lib/**

### D3 JS

D3.js can bind any arbitrary data to a Document Object Model (DOM), and then, through the use of JavaScript, CSS, HTML and SVG, apply transformations to the document that are driven by that data. This library is mainly used in the application for generating geographic graphs using provided data. Please visit the link to read more details <https://d3js.org/>.

### Highcharts JS

Highcharts is a charting library written in pure JavaScript, offering an easy way of adding interactive charts to your web site or web application. Highcharts currently supports line, spline, area, areaspline, column, bar, pie, scatter, angular gauges, area range, areasplinerange, columnrange, bubble, box plot, error bars, funnel, waterfall and polar chart types. This library comes bundled with Wex Appbuilder.

### Angular JS

AngularJS is a JavaScript based open source toolset for building the framework most suited to application development. It is fully extensible and works well with other libraries. Every feature can be modified or replaced to suit your unique development workflow and feature needs. AngularJS follows MVC and MVVM architectural pattern which makes single page application easy to develop and maintain. Following are some of the other libraries used to support AngularJS implementation in this application.

* angular.min.js – Core Angular library.
* Angular-ui-router.min.js – Angular Router library.
* Angular-datamaps.js – Library for geo JSON graph.
* topojson.min.js – Supporting library for USA states and topo graphic data.
* datamaps.all.js – Supporting library for USA states and topo graphic data.
* dirPagination.js – Re-usable angular directive for pagination.
* Underscore-min.js – Re-usable angular directive for drop down.
* Ui-tree.js – Re-usable angular directive for the drop down with tree structure.
* HierarchyNodeService.js – Re-usable service component for drop down and tables
* hierarchySearch.js – Re-usable directive for the search service.
* moment.js – Re-usable date library.
* Moment-fquarter.js – Supporting library for moment.js.
* bootstrap.js – Required Library for bootstrap CSS.
* Ui-bootstrap-tpls-2.5.0.js – Templates for UI bootstrap css elements.
* Angularjs-dropdown-multiselect.min.js – Re-usable directive for multi-select dropdowns.
* daterangepicker.js – Re-usable JS library for date picker calendar.

### Application UI Architecture

The application is designed on the basic MVC pattern using Angularjs and Wex Appbuilder as shown in the below diagram.



Application Architecture Diagram

Brief description of important components of application architecture based on AngularJS.

***Application*** – This is a root module which binds all other sub-models and components together.

***Module*** – Module in AngularJS is like a container. It can contain different parts of the application: Controllers, Services, Filters, and Directives etc. There can be one or more than one modules depending on how the application complexity. The advantages of using Modules in AngularJS are:

* Bootstrapping of the application can be specified by module declaratively and hence they are easy to understand
* Modules are reusable
* Modules can be loaded in any order or even parallel because they delay execution.
* Testing is easier and faster as you need to load only those modules which are needed by the app.

***Config***– This is needed to inject the constants and providers into application configuration blocks. Config block is executed as soon as the module loads. The routing configuration is done in the configuration. It can also be used to configure modules and add HTTP interceptors.

***Routing*** – A large application requires multiple controllers and multiple views – code will not be modular and easy to maintain if the entire application has a single controller or a single view. Routing is used to navigate between different views.

***Controller*** – Controllers contains the logic and share state. Controllers are in-charge of the application and build the models for the views and directives. Multiple controllers can be created for an application. A controller also provides commands for the view to act upon using number of events.

***Views***– A view is the information to render to the end users browser. A view is also called as AngularJS compiled DOM.

***Directives*** - Directives are extensions to HTML. They extend HTML instead of manipulating HTML. AngularJS provides a number of directives out-of-box. For example - ng-app, ng-controller, ng-view, ng-repeat etc. Custom directives can be built.

***$Scope*** – A $scope is the glue between a Controller and a View. The controller generates the Model on $scope. It acts as a View Model for the view.

***Services***– Services play a vital role in AngularJS. It is a component in AngularJS to perform a specific job like:

* Executing a reusable logic which can communicate over HTTP
* Performing computation on an algorithm
* To implement validation checks against data
* To invoke the local store stored in a browser or manipulate cookies
* Act as an abstraction layer between any external resource and application’s components

***Two-Way Binding*** - AngularJS provides two-way data binding out-of-the-box. When model changes, the view reflects the changes without waiting for any specific event to be fired. Also, when a view updates the model, the model gets updated automatically.

***Endpoints***– Endpoints are custom functions that access entities and external web services to deliver data for use by widgets and external systems. These endpoints are invoked by the service layer using Ajax API.

***Entities*** - Each entity equates to a single result delivered into Application Builder from one of the data sources. A result contains information in fields describing the entity instance. Each entity created in the Litigation Billings Insights solution maps to a collection in the WEX Engine.

***WEX Engine Collections -*** Each collection in WEX engine represents the backend data view.

## UI Layouts

Appbuilder provides a built-in layouts for organizing the UI widgets. Appbuilder layout consists of following components.

#### Pages

Pages can be created in the Appbuilder using page creation task.

#### Widgets

Widgets are used to display information to the end user on an application page. Widgets can be created using create widget task. Widgets are created on per page basis. Due to this limitation, we have used angular which allows us to build re-usable components which can be used across number of pages.

#### Page Layout

Page layout option is used to pre-select the organization of widgets in rows and columns. There are multiple layout options available to choose from, but for this application a single row layout is used as angular and bootstrap renders all views using SPA technique.

#### Display

Display option in the settings section of Appbuilder is used to set the header, header meta-data attributes, navigation, footer, custom JavaScript includes and CSS.

## Application Components

A billing insight application is designed and developed using the above artifacts. Following are the details of each components organized in the form of page views and widgets. The master angular application module wexdashboard is loaded and initialized only once for the application, each page can add or remove the components from this app module as on needed basis.

### Navigation Widgets

A common navigation widget contains the top header portion which allows users to navigation from one page to other. Currently the application supports the below listed header links. There are multiple sub-tabs on each page which are link controlled by the tab components and they are not part of the navigation widget. This navigation component contains standard html/css and JS code. Read the code under navigation widget for further details.

* Home
* Invoices
* Firms
* Matter
* Team
* Analysis

### Dashboard/Home View

Dashboard/Home page view is the first page user lands on after login. The page consists of two widgets as listed below. Refer to the functional specification document for details on the prototype and requirements.

#### Dashboard Styles/HTML

This widget contains the custom styles and html specific to dashboard view. The dashboard template html file is located on the server in the folder **/opt/ibm/WEX/AppBuilder/wlp/usr/servers/AppBuilder/apps/AppBuilder/dashapp/src/views/dashboard/dashboardview.html**

#### Dashboard Script

This widget contains the custom JavaScript required to render the dashboard view consisting of components USA/State and County choropleth, bar graph, horizontal bar graph and tables. The script file is located on the server in the folder **/opt/ibm/WEX/AppBuilder/wlp/usr/servers/AppBuilder/apps/AppBuilder/dashapp/src/views/dashboard/dashboardapp.js** and included in the widget.

##### MainController

MainController is responsible for initializing the scope object. It contains various method to call services and directives to create the two way data binding for the HTML components.

##### Service

DataService is responsible for retrieving the backend data asynchronously by invoking the following endpoints.

* Dashboard – Retrieves the backend data related to spending/budget, hours billed, top anomalies, tracked matters and top firms.
* getAnomaly – Retrieves the backend data for top firm anomalies.

##### Directives

Dashboard contains number of directives. These directives are re-usable html elements with customized options to render different datasets.

* Datamaps – This directive is responsible for rendering the usa/state/county map.
* hcBarChart – This directive is responsible for rendering vertical bar charts (spending/budget, Hours Billed and Anomalies) on dashboard widget.
* hcHbChart – This directive is responsible for rendering horizontal bar charts (tracked matters) on dashboard widget.

### Invoices View

This page view shows the information related to invoices. The page consists of two widgets as listed below. Refer to the functional specification document for details on the prototype and requirements.

#### Invoices Styles/HTML

This widget contains the custom styles and html specific to invoices view.

#### Invoices Script

This widget contains the custom JavaScript required to render the invoice view consisting of components invoice lists and invoices details. The script file is located on the server in the folder **/opt/ibm/WEX/AppBuilder/wlp/usr/servers/AppBuilder/apps/AppBuilder/dashapp/src/views/** **invoices/invoicesapp.js** and included in the widget.

##### Controller

InvoiceController is responsible for initializing the scope object. It contains various method to call services and directives to create the two way data binding for the Html components.

##### Service

InvoiceDataService is responsible for retrieving the backend data asynchronously by invoking the following endpoints.

* InvoiceList – Retrieves the data from the backend for all invoices.
* InvoiceListItemList – Retrieves the details of each invoices and line items.

##### Directives

Invoices view contains number of directives. These directives are re-usable html elements with customized options to render different datasets.

* dirPagination – This directive is responsible for rendering the pagination for the invoice list and invoice details view.
* hierarchySearch – This directive is responsible for rendering drop down filters to sort and search the invoices and invoice detail item list.

### Firms View

This page view shows the information related to firms. The page consists of two widgets as listed below. Refer to the functional specification document for details on the prototype and requirements.

#### Firms Styles/HTML

This widget contains the custom styles and html specific to firms view.

#### Firms Script

This widget contains the custom JavaScript required to render the firms view consisting of components firm’s lists and firm’s details. The script file is located on the server in the folder **/opt/ibm/WEX/AppBuilder/wlp/usr/servers/AppBuilder/apps/AppBuilder/dashapp/src/views/** **firms/firms.js** and included in the widget.

##### Controller

FirmsController is responsible for initializing the scope object. It contains various method to call services and directives to create the two way data binding for the Html components.

AccordionCtrl is responsible for controlling the display of the master-detail view with accordion style.

##### Service

FirmDataService is responsible for retrieving the backend data asynchronously by invoking the following endpoints.

* FirmsList – Retrieves the data from the backend for all firms.
* FirmsDetailsList – Retrieves the details of each firm.

##### Directives

Firms view contains number of directives. These directives are re-usable html elements with customized options to render different datasets.

* dirPagination – This directive is responsible for rendering the pagination for the firms list and firm’s details view.
* hierarchySearch – This directive is responsible for rendering drop down filters to sort and search the firms list and firms detail list.
* hcBarChart2 – This directive is responsible for rendering all the bar charts on firms view.
* pfTable – This directive is responsible for rendering the tables on this page.

### Matters View

Matters view shows the information related to matters. The page consists of two widgets as listed below. Refer to the functional specification document for details on the prototype and requirements.

#### Matters Styles/HTML

This widget contains the custom styles and html specific to matters view.

#### Matters Script

This widget contains the custom JavaScript required to render the matters view consisting of components matter’s lists and matter’s details. The script file is located on the server in the folder **/opt/ibm/WEX/AppBuilder/wlp/usr/servers/AppBuilder/apps/AppBuilder/dashapp/src/views/** **matters/matters.js** and included in the widget.

##### Controller

MatterController is responsible for initializing the scope object. It contains various method to call services and directives to create the two way data binding for the Html components.

##### Service

MatterDataService is responsible for retrieving the backend data asynchronously by invoking the following endpoints.

* MattersList – Retrieves the data from the backend for all matters.
* MattersDetails – Retrieves the details of each matter.

##### Directives

Matters view contains number of directives. These directives are re-usable html elements with customized options to render different datasets.

* dirPagination – This directive is responsible for rendering the pagination for the matters list and matter details view.
* hierarchySearch – This directive is responsible for rendering drop down filters to sort and search the matters list and matter detail list.
* hcBarChart2 – This directive is responsible for rendering all the bar charts on matter view.
* hcProgressChart – This directive is responsible for rendering all the progress charts on matter view.
* pfTable – This directive is responsible for rendering the tables on this page.

### Team View

This page view is not built yet.

### Analysis View

This page view allows user to select different filters and render reports using graphs. The page consists of two widgets as listed below.

#### Analysis Styles/HTML

This widget contains the custom styles and html specific to analysis view.

#### Analysis Script

This widget contains the custom JavaScript required to render the analysis view. The script file is located on the server in the folder **/opt/ibm/WEX/AppBuilder/wlp/usr/servers/AppBuilder/apps/AppBuilder/dashapp/src/views/** **analysis/analysis.js** and also included in the widget.

##### Controller

AnalysisController responsible for initializing the scope object. It contains various method to call services and directives to create the two way data binding for the Html components.

##### Service

AnalysisDataService is responsible for retrieving the backend data asynchronously by invoking the endpoints.

* AnalysisDataEndpoint – Retrieves the data from backend for all dropdowns.
* AnalysisCharts – Retrieves the data from backend required to render different charts.
* AnalysisMatterDropDown – Retrieves the data from backend for matters drop down.
* getMatterChart – Retrieves the data from backend for matter specific table chart.

##### Directives

Analysis view contains number of directives. These directives are re-usable html elements with customized options to render different datasets.

* angularjs-dropdown-multiselect – This directive is responsible for rendering all the dropdowns on analysis view.

##### Custom Chart Functions

Analysis view contains number of functions to draw different charts based on the dropdown filter selection. Currently there are three types of charts used on analysis view.

* Barchart – This directive is responsible for rendering all the bar charts on analysis view.
* timelinechart– This directive is responsible for rendering all the timeline charts on this view.
* piechart– This directive is responsible for rendering the pie charts on this page.