BluePrint® SPA

Front-End Architecture

Version: first draft - in progress

Author: Pegah Tabrizi

Table of Contents

[Overview 3](#_Toc431206006)

[Technology Stack 3](#_Toc431206007)

[JavaScript SPA Frameworks: AngularJS & TypeScript 3](#_Toc431206008)

[Frontend Workflow Automation - Bower & Gulp 4](#_Toc431206009)

[Blueprint High Level Components 4](#_Toc431206010)

[Application Administration Views 4](#_Toc431206011)

[Project Administration Views 4](#_Toc431206012)

[Artifact Management Views 4](#_Toc431206013)

[Activity Simulation Views 4](#_Toc431206014)

[Baseline Review Views 4](#_Toc431206015)

[Rapid Review Views 4](#_Toc431206016)

[Impact Analysis Views 4](#_Toc431206017)

[Artifact Management / Main Experience – Layout 6](#_Toc431206018)

[Features of Layout in Current SilverLight Implementation 6](#_Toc431206019)

[Features we are planning to add 6](#_Toc431206020)

[Layout Managers from 3rd Parties 6](#_Toc431206021)

[Artifact Management / Editor Panel – TreeGrid 7](#_Toc431206022)

[Features of TreeGrid in Current SilverLight Implementation 7](#_Toc431206023)

[Nice to have features – We might plan to add 8](#_Toc431206024)

[AngularJs TreeGrids from 3rd Parties 9](#_Toc431206025)

# Overview

With Microsoft planning on retiring Silverlight in near future and browsers starting to drop their support for Silverlight already, we have to adopt open web standards such as HTML5/JavaScript and replace Blueprint’s current offering that had been developed using Silverlight and along the way add support for tablet and mobile.

A Web based single page application (SPA) is an evolution away from the stateless page-redraw model that browsers were originally designed for but advancements in HTML5 and new JavaScript frameworks, enable browser to retain a single page and create a fluid user experience similar to a desktop application even when the application requires server communication.

Modern browsers that can parse HTML5 allow developers to shift the user interface and related application logic from web servers to the client and JavaScript libraries such as AngularJS, ReactJS, Ember, Meteor, … have adopted single page application (SPA) principles. HTML5 web-sockets also provide a bidirectional real-time client-server communication when needed.

Because SPA architecture moves the logic from the server to the client, the role of the web server can also evolve into a pure data API or web services enabling the server to be built as reusable discrete polyglot micro-services that are highly scalable, highly available and can independently be deployed in cloud environments.

# Technology Stack

## JavaScript SPA Frameworks: AngularJS & TypeScript

With AngularJS popularity and a large open source community behind it, not to mention the backing of Google, Angular has also been Blueprint’s JavaScript framework of choice. There is some uncertainty however with next version of Angular being completely different than Angular 1.x and because of it we need to design our code in a way that we are prepared for the changes which can be categorized into the following:

* Template Syntax
* ES6 and Modules
* Types
* Components
* Bindings

In Angular 2, “components” are the main way elements and logic are built on the page, and Angular 1.x’s directives, controllers, and scope are all combined into components. Using TypeScript with Angular 1.x provides a more clear migration path to Angular2.x

TypeScript is superset of ES5 and it wraps ES6 so we can use all ES6 features and on top of that it adds types and annotations that come very handy in large projects such as ours and can improve JavaScript development experience by providing type information and displaying API documentation (“intellisense”) based on type definition files provided by “Definetely Typed” repo. Using TypeLite we can also create TypeScript definitions (interfaces/enums) from our .NET classes.

TypeScript is transpiled into JavaScript so our workflow need to include this step. We also need write our controllers and services as TypeScript classes.

Using TypeScript decorators is a neat feature that Angular 2 uses which could make Angular1 code look like Angular2 but I think more than worrying about the looks and syntax, we need to use an architecture that is aligned with Angular2 which means no $scope anywhere and fully componentizing our app and thinking more about how components communicate with each other.

Looking into Functional Reactive Programming / React&Flux&Immutable objects and using React.js with Angular could be also something that worth looking into.

## Frontend Workflow Automation - Bower & Gulp

TODO – frontend technology stack / tools / workflows

# Blueprint High Level Components

Figure 1 - Shows the breakdown of major components in blueprint web application:

## Application Administration Views

Application Administration module contains views where application administrator can setup and manage application instance. All application settings are managed from the views and dialogs of Application Administration module.

## Project Administration Views

Project Administration module contains views where project administrator can setup and manage project. All project settings are managed from the views and dialogs of Project Administration module.

## Artifact Management Views

Artifact Management module contains views where users can create, update, manipulate and delete artifacts, its properties and alike. All textual and graphical representations and editors of the artifacts are implemented in Artifact Management module.

## Activity Simulation Views

Activity Simulation module contains views where user runs screen mockup and use case simulations.

## Baseline Review Views

Baseline Review module contains views where user can define baseline and create review of artifacts as of baseline date.

## Rapid Review Views

Rapid Review module contains views where user can review collection of artifacts.

## Impact Analysis Views

Impact Analysis module contains views where user can trace and explore artifact dependencies.



Figure 1 – Blueprint Web Application High Level Design

# Artifact Management / Main Experience – Layout

Modern user interfaces make extensive use of multiple windows, panels, tabs and views that enable users to explore different pieces of information and solve different tasks. Efficient usage of coordinated and multiple views requires a window management framework that allows users to interactively arrange their views within the available screen-space by using features such as:

* Resizable/Fixed size panels
* Collapsible/Expandable panels
* Docking / Undocking (Floating dialogs)
* Pinning / Unpinning
* Tabbed panels – dynamic tab creation
* Maximizable / Minimazable
* Responsive design
* Persistable – be able to save the state and reload it
* Drag & Drop
* Hidable panels
* Slidable panels

A variety of software libraries have been developed to support window management for web-based interfaces. Examining all of these libraries is a tedious task for software developers, who need to select an appropriate library for their purpose without the need for extensive tests and experimental implementations.

## Features of Layout in Current SilverLight Implementation

* Resizable and fixed size panels
* Collapsible / Expandable panels with title when collapsed
* Tabbed panels – dynamic tab creation
* Persistable

## Features we are planning to add

* Docking / Undocking (Floating dialogs)
* Pinning / Unpinning
* Maximizable / Minimazable
* Responsive design
* Slidable panels
* Hidable panels (nice to have)
* Drag & Drop (nice to have)

TODO – expand on feature set

## Layout Managers from 3rd Parties

Here are a number of layout manager libraries from third parties

* Golden Layout <https://golden-layout.com/>
* Web Cabin Docker <http://docker.webcabin.org/>
* DockSpawn <http://www.dockspawn.com/>
* jQuery UI Layout <http://layout.jquery-dev.com/>

The following widget libraries also provide a “splitter” widget but leave the management of layout to the developers – which means more coding for us:

* Wijmo splitter

<http://wijmo.com/widgets/wijmo-open/splitter/>

* Kendo UI splitter

<http://demos.telerik.com/kendo-ui/splitter/index>

* jqWidgets splitter

<https://www.jqwidgets.com/jquery-widgets-documentation/documentation/angularjs/angularjs-directives/angularjs-jquery-splitter.htm>

TODO - compare feature set / pro-cons and decide which 3rd party widget to use

# Artifact Management / Editor Panel – TreeGrid

A hierarchical tree grid is currently used in Blueprint’s SilverLight editor panel to show:

* Project contents
* Baseline and Reviews
* Collections

Based on the feedback we received from PMs, our customers are happy with current set of its features and we should minimize introducing too many changes and do not cut any feature that they currently have.

## Features of TreeGrid in Current SilverLight Implementation

* Column Sorting
  + Internal Sorting (client side)
  + External Sorting (server side)
* Column Filtering
  + Internal
  + External
* Column Width Adjustment
* Dragging
  + Column dragging
  + Row dragging
    - Single Row
    - Multi-Row
* Row Selection
  + Single Selection / to choose context
  + Multi-Selection
  + Checkbox Selection on first Column
* Cell Editing / Cell Selection
* Hierarchical / TreeGrid
* Tooltip
  + Table Cell
  + Header Cell
  + While dragging rows
* Context Menu
  + On Row
  + On Header
* Show icons in cell
* Show RichText in cells
* Dynamically Add Row
  + Same Level
  + Child Level
* Dynamically Add Column
* Cut/Copy/Paste
  + Single Row
  + Multi Rows
* i18n support
* Keyboard Navigation
* Grid Scrolling
  + Fixed Header
* Large Data Sets / Lazy Loading of grid
  + On node expansion
  + On scroll
* Scroll
  + Infinite Scrolling
  + Pagination (?)
* Auto Resizing
* Two way data binding

TODO – expand on feature set

## Nice to have features – We might plan to add

* Pinning
  + Frozen column
  + Frozen Rows
* Quick Search/Filter – multi column
* Column Grouping with expand/collapse

TODO – expand on feature set

## AngularJs TreeGrids from 3rd Parties

* ui-grid / ng-grid <http://ui-grid.info>
* ag-grid: <http://www.angulargrid.com/>
* wijmo5 grid: <http://demos.componentone.com/wijmo/5/Angular/Explorer/Explorer/#/grid/intro>
* jqxgrid: <http://www.jqwidgets.com/jquery-widgets-demo/demos/jqxgrid/index.htm>
* Kendo-UI grid: <http://www.telerik.com/kendo-ui/grid> (doesn’t support hierachical tree)

TODO - compare feature set / pro-cons and decide whether to write our own tree-grid from scratch or use one of these 3rd party widgets.