Blueprint Web Application a.k.a Nova

Requirements / High-level Design / Architecture

… My Thoughts

Version: 1.0

Author(s): Pegah Tabrizi

Table of Contents

[Overview 3](#_Toc440557716)

[Technology Stack 3](#_Toc440557717)

[JavaScript SPA Frameworks: AngularJS & TypeScript 3](#_Toc440557718)

[User Interface Requirements 4](#_Toc440557719)

[Layout 4](#_Toc440557720)

[Features of Layout in Current SilverLight Implementation 4](#_Toc440557721)

[Features we are planning to add 4](#_Toc440557722)

[Layout Managers from 3rd Parties 5](#_Toc440557723)

[TreeGrid for EditorPanel / Tree for Explorer 5](#_Toc440557724)

[Features of TreeGrid in Current SilverLight Implementation 5](#_Toc440557725)

[Nice to have features – We might plan to add 6](#_Toc440557726)

[AngularJs TreeGrids from 3rd Parties 7](#_Toc440557727)

[Other JS Libraries 7](#_Toc440557728)

[Frontend Workflow Automation - Bower & Gulp 7](#_Toc440557729)

[Bower 7](#_Toc440557730)

[Gulp 8](#_Toc440557731)

[Nova prototype 9](#_Toc440557732)

[Using KendoUI 10](#_Toc440557733)

[Using open-source components such as jquery-layout, bootstrap, jstree 10](#_Toc440557734)

[Using a bootstrap admin template 11](#_Toc440557735)

[Using jqWidgets 12](#_Toc440557736)

# 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.)

# User Interface Requirements

## 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)

### 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/>
* jQWidget Docking Layout   
  <http://www.jqwidgets.com/jquery-widgets-demo/demos/jqxdockinglayout/index.htm#demos/jqxdockinglayout/defaultfunctionality.htm>

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>

## TreeGrid for EditorPanel / Tree for Explorer

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. Here are the list of some high-level features that are supported in SilverLight version:

### 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

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

* Pinning
  + Frozen column
  + Frozen Rows
* Quick Search/Filter – multi column
* Column Grouping with expand/collapse
* Built-in OData support

### 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://demos.telerik.com/kendo-ui/treelist/index>
* Others: DevExtreme Web, Syncfusion HTML5 controls, Infragistics HTML5 controls, OpenUI5, JayData, Breeze.js, datajs, ODataJS, angular-odata
* …

## Other JS Libraries

We’ve already decided to use the following 3rd party libraries:

* mxGraph – as diagram library (currently in use in RapidReview)
* d3 – for light-weight graphing (currently in use in ImpactAnalysis)
* TinyMCE – for WYSIWYG HTML editing (currently in use in RapidReview)

It’s been also decided not to use KendoUI going forward and mainly rely on Bootstrap.

For any other widgets that is not provided by bootstrap, we might want with a third party library from same vendor we are picking the “Layout” or “TreeGrid” to minimize the risk of layout/library incompatibilities.

As for non-widget utility libraries we are currently using bower.js for browser detection, and probably would like to include modernizr for browser feature detection. Also possibly a utility library for javascript development such as lodash or underscore.js

# Frontend Workflow Automation - Bower & Gulp

Moving towards becoming independent of IDE, bower and gulp is to be used for automation of frontend workflow.

## Bower

Bower is a package manager primarily for front-end development. It needs to be installed using npm (node package manager).

npm install -g bower

I’ve included an initial version of bower for Nova prototype

<https://github.com/BlueprintSys/blueprint/blob/develop/app/Blueprint/bower.json>

To install bower packages please run

bower install

My initial version of bower includes dependencies to libraries that were in use (in RapidReview & ImpactAnalysis) as we started Nova prototype namely:

* Angular > 1.4.1 (Angular-mocks / Angular-ui-router/ …)
* Bootstrap >3.3.5
* jQuery > 2.1.1
* d3 (for ImpactAnalysis)
* fontawesome
* jquery-ui (? – might be able to remove this)
* jobx
* tinymce (for RapidReview)
* ui-select
* kendo-ui-core (? – might be able to remove this)

But also added a

* jstree (a popular jquery library for building trees with lazy loading features)
* jqwidgets (mainly because of a docking/pinning feature in jqxDockingLayout)

Please note for non-open-source libraries (such as mxgraph), we can’t use bower and we need to include the library separately.

A notable alternative to bower, worth looking at is **jspm** to enable us, dynamically load javascript libraries as needed.

## Gulp

Gulp is used to automate build process for front-end development. It needs to be installed using npm (node package manager).

npm install -g gulp

I’ve included an initial version of gulp for Nova prototype here:

<https://github.com/BlueprintSys/blueprint/blob/develop/app/Blueprint/gulpfile.js>

Run

gulp help

to see the list of available tasks:

bower [start-build]

build [post-build]

clean

content [start-build]

css [start-build]

default Display this help text. [help]

end-build [bower, js, css, content]

help Display this help text.

js [ts]

post-build [end-build]

run [build]

serve

start-build [clean]

ts [start-build]

Nove prototype was developed without using TypeScript however in order to make sure we can compile typescript using gulp a task had been developed to do so. Run

gulp ts

Also I had started playing with integrating gulp with Visual Studio using its TaskRunner but at the time of this writing I didn’t get to finish it all as my main development for Nova prototype was agnostic to the IDE (I used sublime and ran commands through git bash) but there are tasks there to minify and concatenate javascript and css.

Another task we need to add to gulp is compiling sass

Run

gulp serve

to start a light-weight node webserver for Nova prototype – there are no server components just mock objects that need to be served from a webserver for nova to work.

# Nova prototype

A working prototype of Nova is currently committed in blueprint repo under **app/Blueprint** directory / develop branch which also includes Scenario-Manager/Storyteller:

<https://github.com/BlueprintSys/blueprint/tree/develop/app/Blueprint>

Instructions on how to run Nova prototype is provided as a README on same page. After running gulp serve, you’d see the prototype on <http://localhost:8000>

Also in order to decide between different layout libraries a number of prototypes were developed currently under

<https://github.com/BlueprintSys/blueprint/tree/develop/prototype>

You can run these prototypes similarly using same commands but from prototype directory and on <http://localhost:9000> you’d see a link to all other prototypes

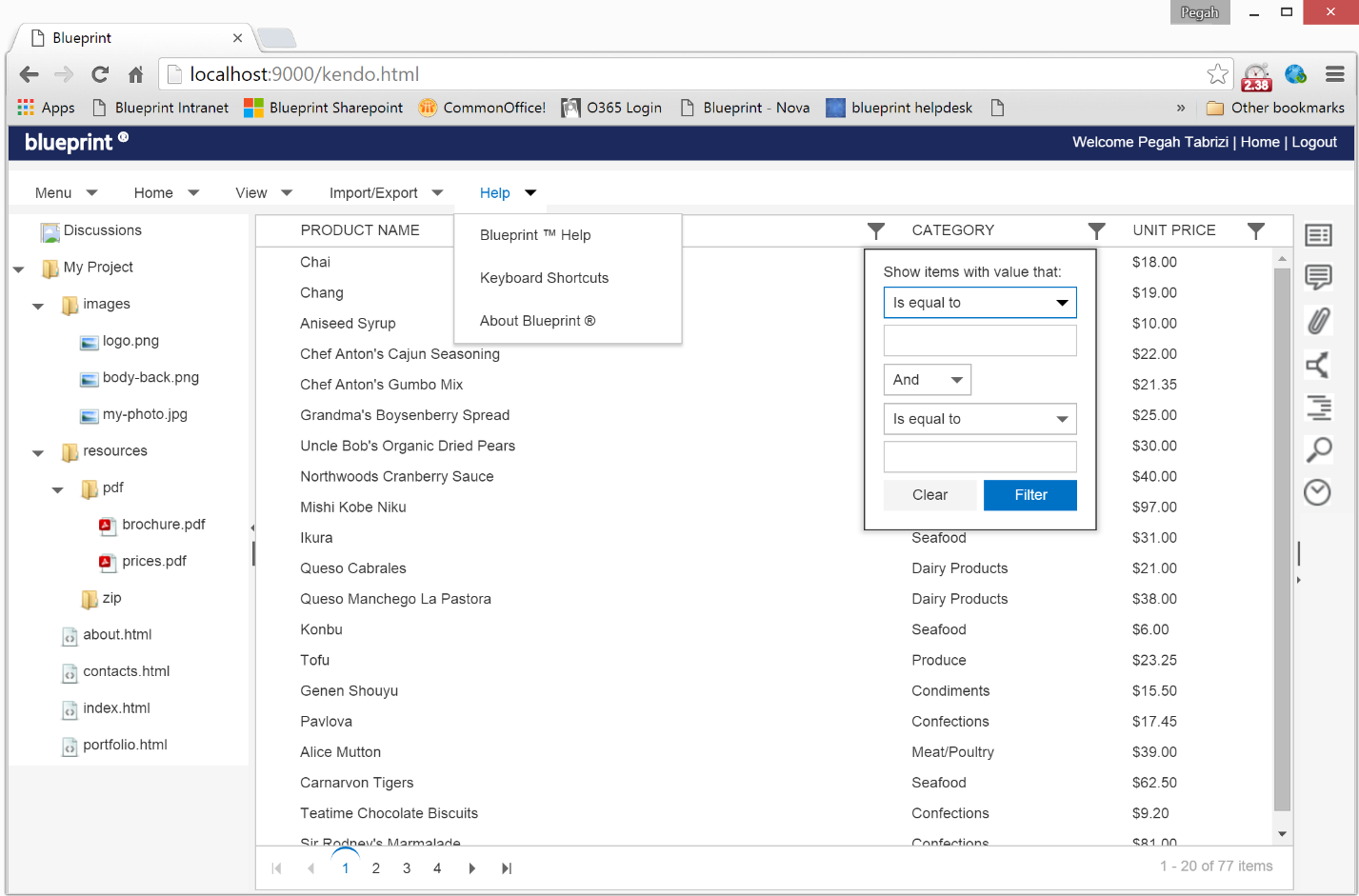
Please note, at the time of developing of most of the prototypes, UX team were also working on their designs in parallel and the main goal of the prototypes were to help us select the widget libraries that support all the features we talked about as well as features currently supported in Silverlight. As a result the look and feel of the prototypes currently does not necessarily match UX team designs and are more in line with the current Silverlight version.

## Using KendoUI

KendoUI is from the same company (Telerik) that developed same widgets for silverlight version so it made sense to stick with the same library vendor in hopes to get similar usability experince.

Also KendoUI was already in use in RapidReview & ImpactAnalysis.

In the prototype, I used their tree, drop-down menu, table, tabs and splitter. However layout dockablity/pinning is missing from KendoUI library as a widget – could be implemented though using splitter but might be too much work…



Independently team decided that they want to stop using KendoUI for Nova

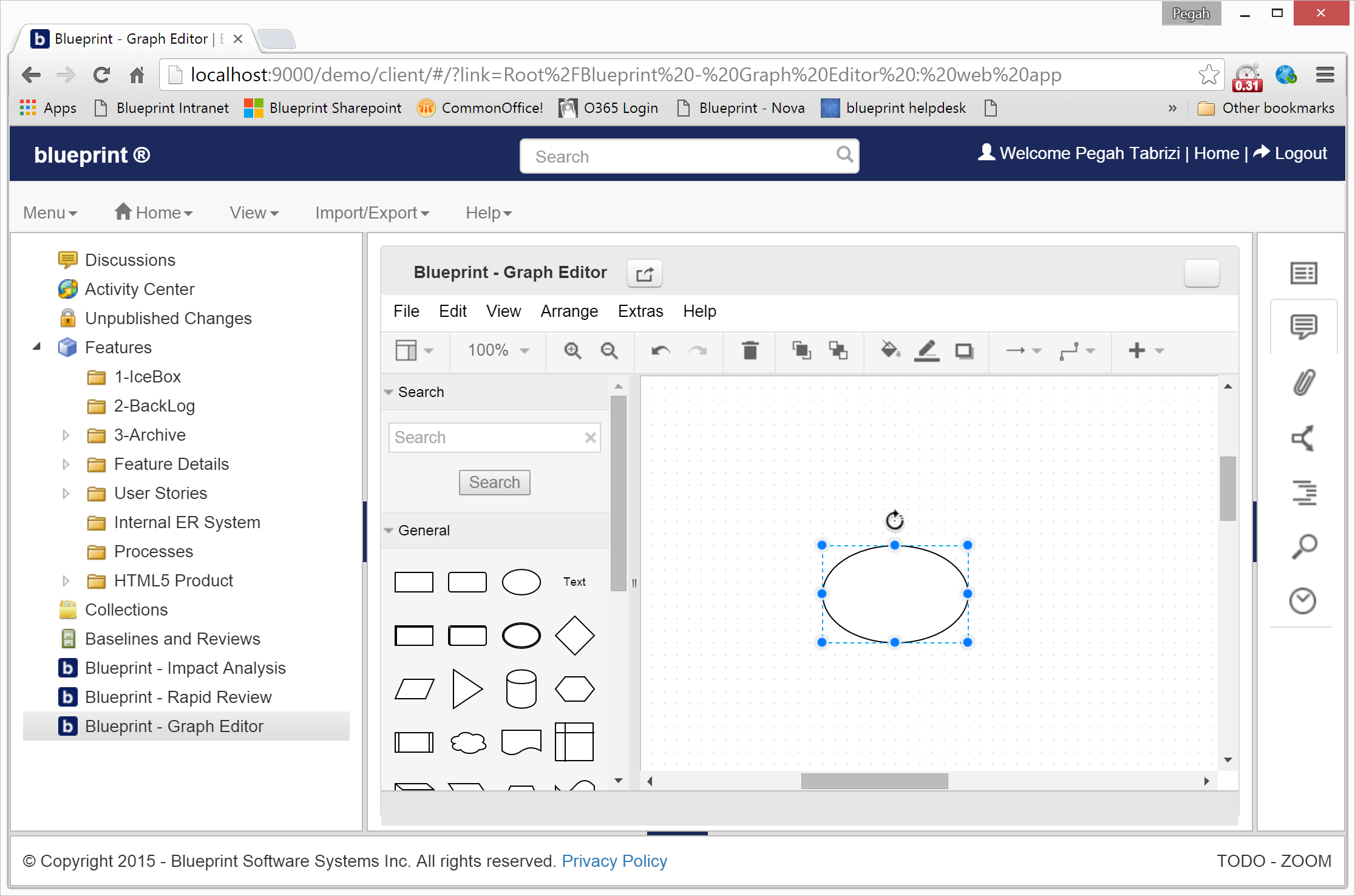
## Using open-source components such as jquery-layout, bootstrap, jstree

As part of this prototype, I wrapped jstree (<https://www.jstree.com/>) as an angular directive. jstree is widely used and supports lazy loading of tree nodes as we expand them.

It also supports drag&drap, context menu and editing/creating nodes dynamically.

I also wrote the code so that on click of each node, we show the contents corresponding to the nodes as boostrap table.

I’ve also tried loading mxgraph editor example, Impact Analysis and Rapid review, inside the main pane on the screen. See blow:



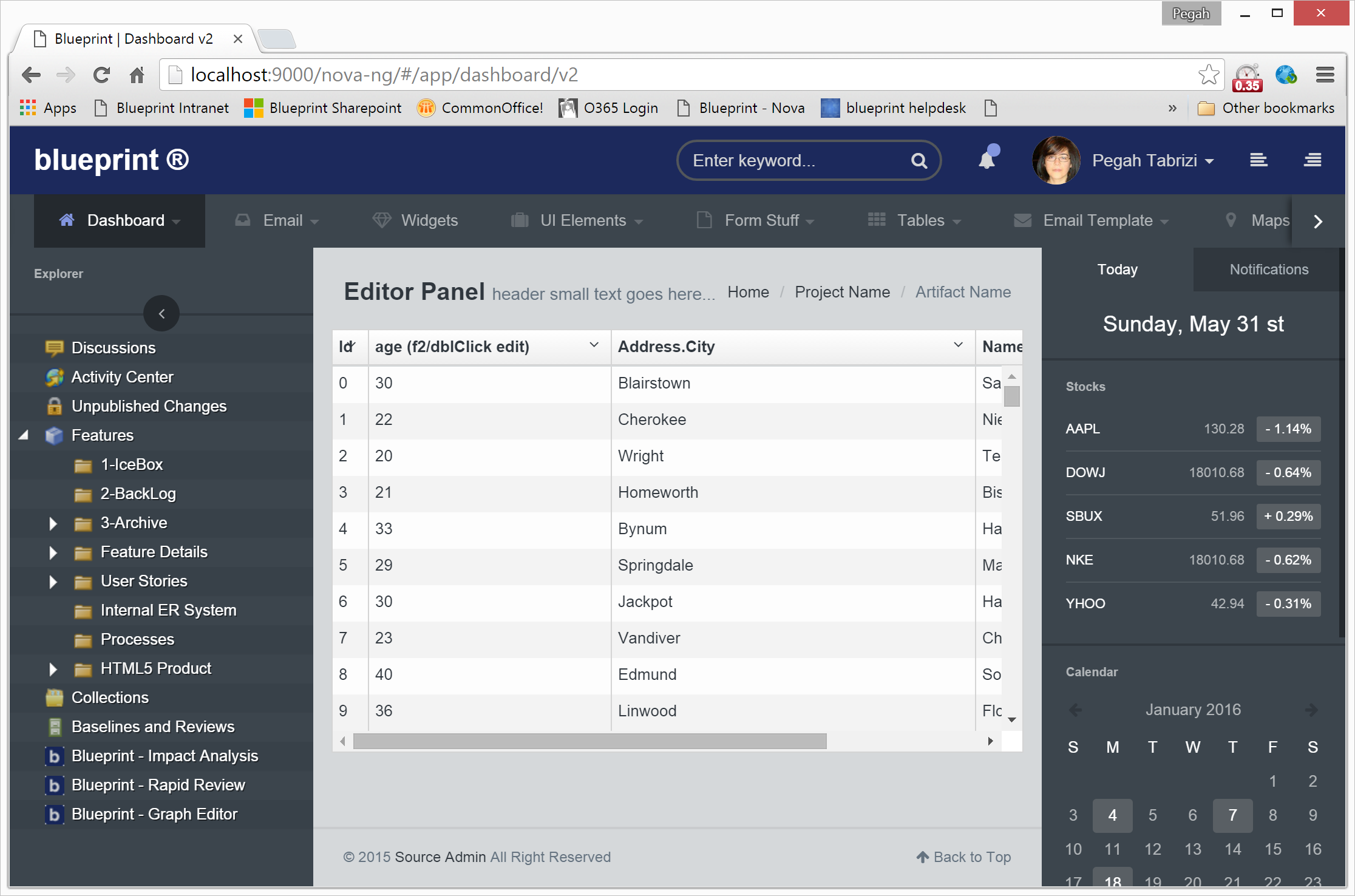
The feature I missed from this prototype was being able to dock/undock, pin/unpin different panels.

## Using a bootstrap admin template

Because UX team’s mockups looked very similar to an admin panel templates available online to purchase and it was unlikely to receive css/icons/… that I needed, I went ahead and purchased a $20 template online from wrapbootstrap.com, called <http://seantheme.com/source-admin-v1.2/admin/html/index_v2.html> (please note the enterprise license is more expensive if we decide to go this route)

And I attempted to customize it but found it is harder to customize. Also the side panel was fixed size.

I was able to customize the explorer tree with jstree and the top panel bar.



## Using jqWidgets

This prototype is using jqWidgets’ (<http://www.jqwidgets.com/>) jqxDockingLayout, jqxMenu, jqxtree, jqxgrid, filter,… and was the only library that I find can satisfy the docking/pinning requirement.

I was able to easily integrate mxgraph and my jstree implementation and I find it easily customizable.

Here’s how it looks for Scenario Manager if we were to open two instances of scenarios together:

