## Overview

A prototype with a test version of the ag-grid component configured as tree is available in the **blueprint** [repository](https://github.com/BlueprintSys/blueprint), under [/prototype/client/ag-grid](https://github.com/BlueprintSys/blueprint/tree/develop/prototype/client/ag-grid)

The prototype can be browsed locally by following this procedure:

* Open a terminal on the directory were local **develop** branch is stored
* Navigate to the **prototype** subfolder
* Run gulp serve to start the local server
* Open the URL <http://localhost:9000/> in a browser
* Click on the [Nova shell prototype 2016](http://localhost:9000/ag-grid) link

The test page includes:

* The tree in the left sidebar
* The main content section (just an empty space where nodes from the tree can be drag&dropped)
* The right sidebar with the full-height accordion
* Custom scrollbars for the tree and utility panel’s components

The sidebars can be opened/closed clicking on the links in the header.

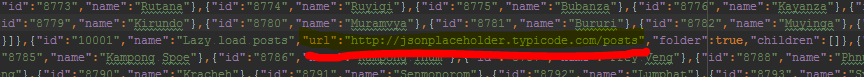
## Accordion/Utility panel

The accordion is almost entirely CSS controlled. I wrote a small directive just to make it full-height and to equally divide the space amongst the opened elements (that behavior cannot be achieved by CSS alone).

## Tree/ag-grid

The tree is initially populated by ~7,500 nodes stored in data.json

Two of the tree nodes support lazy-loading of their subnodes. These two nodes are highlighted in yellow. The lazy-loading happens onclick on the row (please refer to the rowClicked method in the exampleCtrl controller in **app.js**), and uses the endpoint defined in the row data model



The second of the two nodes also allows for nested lazy-loading. Please note that the code used in the test page is for demo purpose only and shouldn’t be used for the final product.

The way ag-grid works doesn’t allow the direct “injection” of new nodes in the HTML structure. Instead, the nodes must be added to the pre-existent data and then the new/modified data must be passed in its entirety to the grid (e.g. $scope.gridOptions.api.setRowData(rowData);). This will automatically issue a redraw of the grid/tree. Since component draws only the rows currently visible in the browser window, the redraw is actually pretty fast and doesn’t seem to cause any visual artifact.

Before redrawing the tree, though, we may want to update the rows data to store which nodes were currently opened, so the when redrawing the tree, the component will re-open them for us. To do that, the data.open property of the node must be set to true (e.g. node.data.open = node.expanded; The expanded property tells if the tree branch had been opened by the user, while data.open is the initial state of the node).

The component doesn’t natively support drag&drop. I added a directive, derived from [here](https://blog.parkji.co.uk/2013/08/11/native-drag-and-drop-in-angularjs.html) as a demo. The demo allows to drag a node from the tree to the empty space in the middle of the page. What is missing from the demo is drag&drop inside the tree itself. Because of the way the component is designed, dragging&dropping a node inside the tree will have to:

* Recognize if the node is going to be dropped inside another node or between two adjacent nodes (sort)
* Make the relevant changes to the rows data model (e.g. move a node and all its children in the children property of the target node)
* Redraw the tree by passing the updated rows data model

Other issues I noticed while testing ag-grid include:

* In Internet Explorer, scrolling quickly through a long list of nodes can cause lags after all the nodes cached in memory have been shown. To mitigate the problem, we can increase the gridOptions.rowBuffer parameter (e.g. 200);
* Unfortunately, ag-grid uses IDs to identify certain HTML elements and that can cause problems if we happen to use the same IDs (“center”, “borderLayout\_eRootPanel”, “borderLayout\_eGridPanel”, …). This is a [known issue](https://github.com/ceolter/ag-grid/issues/260).
* It is possible to have multiple ag-grid components on the same page (i.e. the project explorer and the artifacts grid) as long as we define unique gridOptions. Example:  
  <div ag-grid="gridOptionsProjectExplorer" />  
  …  
  <div ag-grid="gridOptionsArtifactsGrid" />  
  …  
  $scope. gridOptionsProjectExplorer = {

columnDefs: columnDefsProjectExplorer,

rowData: rowDataProjectExplorer,  
…  
$scope. gridOptionsArtifactsGrid = {

columnDefs: columnDefsArtifactsGrid,

rowData: rowDataArtifactsGrid,  
…

## Custom scrollbars/angular-perfect-scrollbar-2

We are using angular-perfect-scrollbar-2 (<https://www.npmjs.com/package/angular-perfect-scrollbar-2>), a wrapper for perfect-scrollbar (<https://github.com/noraesae/perfect-scrollbar>).

perfect-scrollbar is a minimalistic but well regarded (~3,350 stars on GitHub) scrollbar plugin working with jQuery or vanilla JS as well. From its Github page:

**What does perfect mean?**

* There should be no css change on any original element.
* The scrollbar should not affect the original design layout.
* The design of the scrollbar should be (nearly) fully customizable.
* If the size of the container or the content changes, the scrollbar size and position should be able to change.
* New! It should work with vanilla JavaScript and major tools like NPM or Browserify.

**Then perfect-scrollbar is really perfect?**

* perfect-scrollbar has some requirements, but doesn't change or add any style on original elements.
* perfect-scrollbar is designed not to have width or height. It's fixed on the right and bottom side of the container.
* You can change nearly all css styles of the scrollbar. The scrollbar design has no dependency on scripts.
* perfect-scrollbar supports an 'update' function. Whenever you need to update the size or position of the scrollbar, just update.
* Additionally, perfect-scrollbar uses 'scrollTop' and 'scrollLeft', not absolute positioning or something messy.
* perfect-scrollbar supports RTL perfectly on both WebKit and Gecko based browsers.

Notes:

* I encountered some rendering artifacts in IE11 when using the custom scrollbar with the ag-grid component in treeview. I tweaked the tree’s CSS a bit and the artifacts have disappeared from my tests (see lines 121-135 of index.html). An additional backup fix is to make the scrollbar container as position:fixed and position it manually on the page, but this approach is not advisable as it may make harder to have a really responsive layout.
* It is possible to make the scrollbar always visible, instead of showing it on hover only (see lines 136-140 of index.html).
* It is advisable (UX agrees) to use the custom scrollbar on desktops only and disable it for tablets. I included a custom directive not-on-mobile to compile angular-perfect-scrollbar-2 only on desktops. This directive will need to be changed to use the **bowser** object when integrated in Nova, instead of relying on a regex test to detect tablets.
* The angular wrapper currently wraps v0.6.3 of perfect-scrollbar. Latest version is v0.6.10

Other custom scrollbar candidates were:

* ng-scrollbars (<https://github.com/iominh/ng-scrollbars>)   
  Very customizable (see examples at <http://iominh.github.io/ng-scrollbars/>), but has jQuery dependency
* angular-perfect-scrollbar (<https://github.com/itsdrewmiller/angular-perfect-scrollbar>)  
  Another wrapper for perfect-scrollbar, but based on a very outdated version
* ng-scrollbar (<https://github.com/asafdav/ng-scrollbar>)   
  Pure angular component, not very compatible with mobile devices (that may not be an issue, since we would probably disable it on tablets), low(er) rating (just 168 stars on GitHub), and not actively developed
* ngTinyScrollbar (<https://github.com/yads/ngTinyScrollbar>)   
  Pure angular component, doesn't properly support fluid/non-static dimensions (needed in the utility panel, for example)