FDC323665-L

Learning Lab: Your first Forge Dashboard

Jaime Rosales D.
Autodesk, Inc.

Learning Objectives

- Discover the basics of Autodesk Forge
- Learn how to create your first app based on Forge
- Learn how to connect external sources of data into a Forge-based application
- Learn how to build a web application interface for a dashboard

Description

Let's bring data together! In this class, we'll build a dashboard with data from different sources, using Forge as the basis for the interface. This class requires basic knowledge on .NET or Node.js. Our CAD models contain a lot of data available to us. In this lab you will get the hands-on experience of building your Forge application with a visualization of data using 3rd party Charts library. Let's build interactive graphs to boost our forge application.

Speaker(s)

Jaime Rosales is a Senior Developer Advocate with Autodesk Forge. He's part of the team that helps partners/customers create new products and transition to the cloud with the use of Autodesk's new Platform - Forge.

He joined Autodesk in 2011 through the acquisition of Horizontal Systems the company that developed the cloud-based collaboration systems—now known as BIM 360 Glue the Glue. He was responsible for developing all the add-ins for BIM 360 Glue, using the API's of various AEC desktop products.

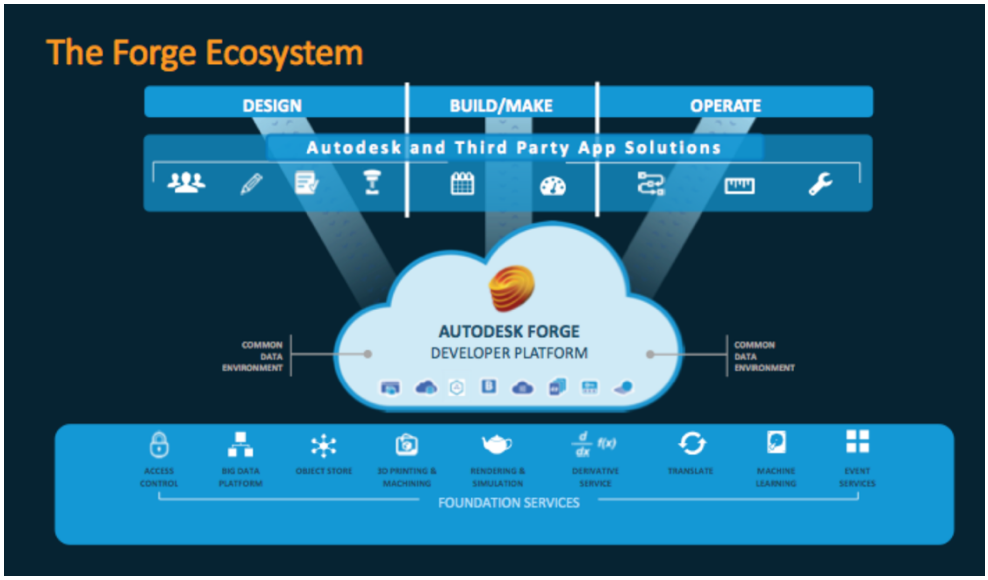
He is currently empowering customers with the use of Autodesk's Forge platform throughout the world, with hosted events such as Forge Accelerators, AEC Hackathons, VR & AR Hackathons. He has been recently involved in the development of AWS Quick Start to support Forge & BIM 360 Applications.

Discover the basics of Autodesk Forge

The Forge platform unlocks the power of design and engineering data so you can connect teams, workflows, and build new services to address today's connected customer.

Forge enables companies to leverage design and engineering data to develop custom software applications and connected workflows for manufacturing, media/entertainment, architecture, engineering, and construction.

- **View 3D models directly in your browser**: The Viewer allows you to embed, interact with, and retrieve meta data about design files in over 50 formats right in your browser, without having to install extra software.
- **Centrally manage data**: The Data Management API allows you to access your data across A360, Fusion, and the Object Storage Service.
- **Convert your design files to formats that unlock their potential**: Use the Model Derivative API to prepare files for the Viewer, extract geometries, retrieve metadata, and more, from over 50 different industry-standard formats.

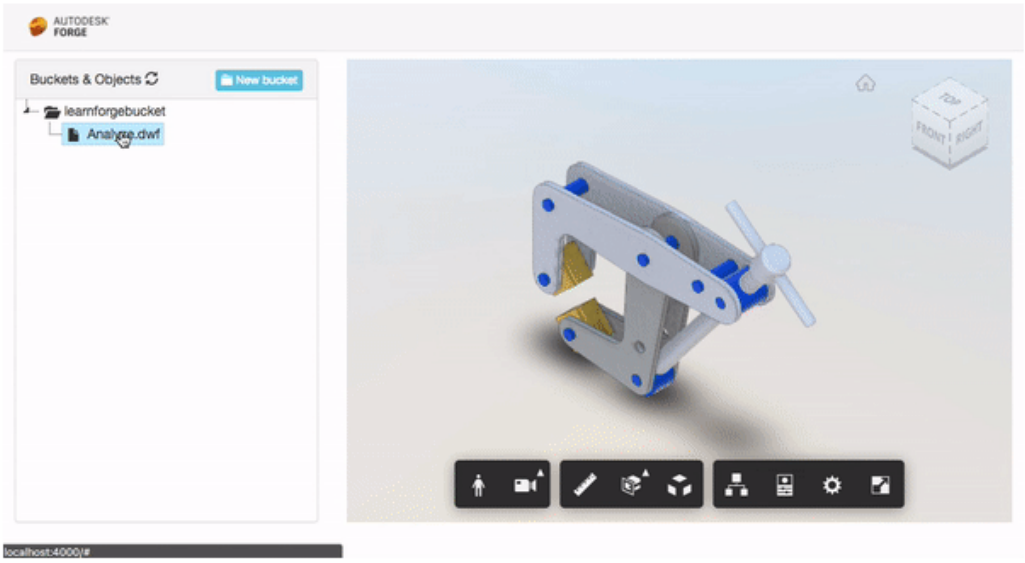


The Forge Ecosystem

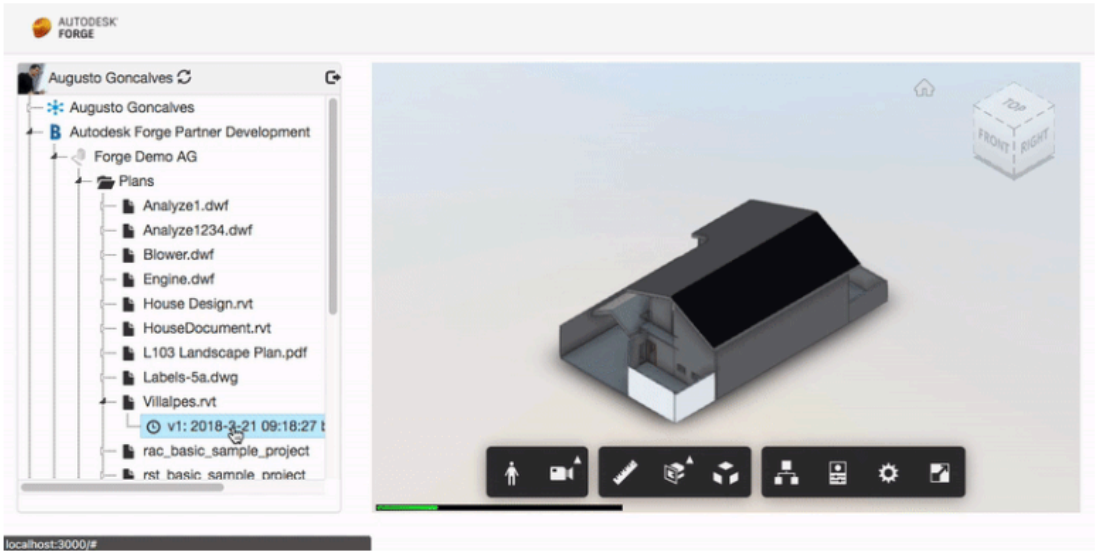
Learn how to create your first app based on Forge

As a pre-requisite it is recommended to attend a couple of earlier Labs that will be future at DevCon this year. Both labs will give you the right based of a project to build from your Dashboard visualization application.

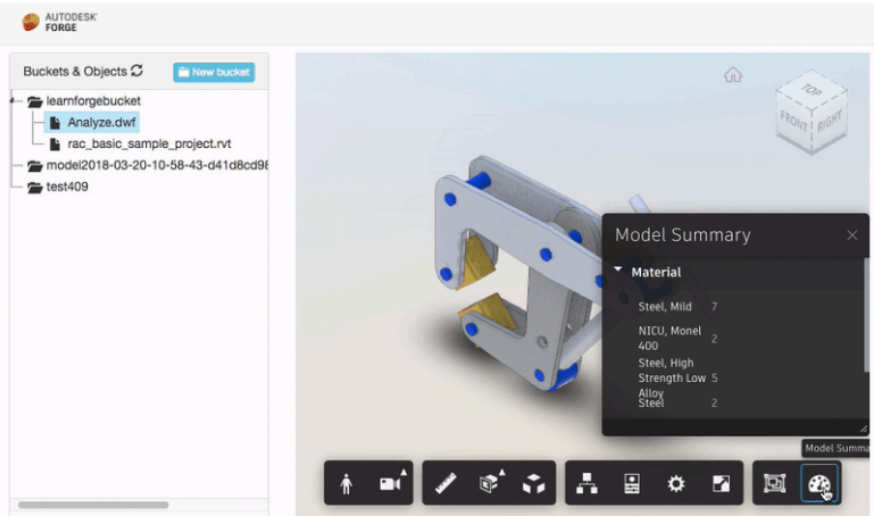
- Learning Lab: Upload and view your models with Forge



- Leaning Lab: Connect to Fusion Team & BIM 360 data



- Learning Lab: Extend the Viewer



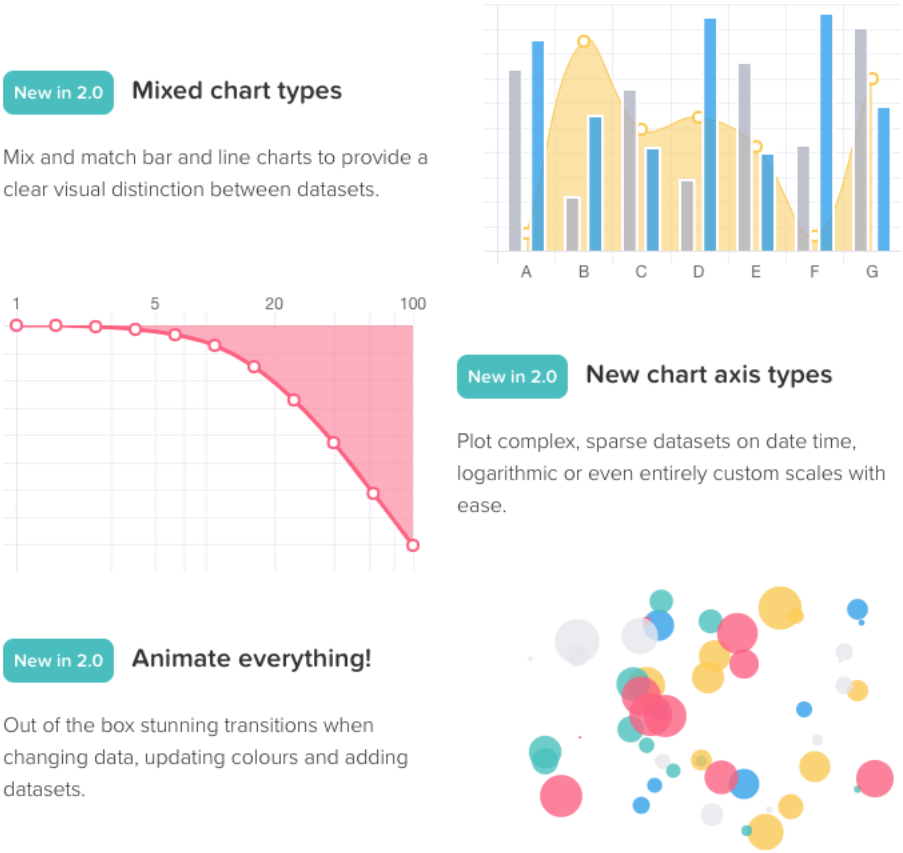
If you don't have a chance to join, you can always try it for yourself through our step-by-step tutorial hosted in learnforge.autodesk.io

This tutorial will walk you through creating your first set of API keys, to later learn the different kind of authentications needed to access the different Forge API services.

Learn how to connect external sources of data into a Forge-based application

Forge being a developer platform allows us to connect to other 3rd party API's to do a combination of technologies and create marvelous applications with good focus into the workflows that we need. In the case of this lab we will use a 3rd party JavaScript library to help us with the representation of the metadata of our CAD model.

ChartJS is a simple yet flexible JavaScript charting for designers & developers.



Learn how to build a web application interface for a dashboard

We will modify the projects mentioned above with the following structure to include the use of charts and a nice UI/UX for being a seamlessly production ready application you can share with your customers.

We will create a new Dashboard folder to treat the new content as extension to the Forge Viewer.

This code will adjust the page layout, watch the Viewer and load the charts when the model date is loaded. It uses JavaScript classes.

```javascript
$(document).ready(function () {
    $(document).on('DOMNodeInserted', function (e) {
        if ($(e.target).hasClass('orbit-gizmo')) {
            // to make sure we get the viewer, let's use the global var NOP_VIEWER
            if (NOP_VIEWER === null || NOP_VIEWER === undefined) return;
            new Dashboard(NOP_VIEWER, [
                new BarChart('Material'),
                new PieChart('Material')
            ])
        }
    });
})

// Handles the Dashboard panels
class Dashboard {
    constructor(viewer, panels) {
        var _this = this;
        this._viewer = viewer;
        this._panels = panels;
        this.adjustLayout();
        this._viewer.addEventListener(Autodesk.Viewing.GEOMETRY_LOADED_EVENT, (viewer) => {
            _this.loadPanels();
        });
    }

    adjustLayout() {
        // this function may vary for layout to layout...
        // for learn forge tutorials, let's get the ROW and adjust the size of the
        // columns so it can fit the new dashboard column, also we added a smooth transition
        var row = $(".row").children();
        $(row[0]).removeClass('col-sm-4').addClass('col-sm-2 transition-width');
        $(row[1]).removeClass('col-sm-8').addClass('col-sm-7 transition-width').after('<div c
    }

    loadPanels = function () {
        var _this = this;
        var data = new ModelData(this);
        data.init(function () {
            $('#dashboard').empty();
            _this._panels.forEach(function (panel) {
                // let's create a DIV with the Panel Function name and load it
                panel.load('dashboard', viewer, data);
            });
        });
    }
}
```

At the **index.html** add a <script> for this new file. This should go inside the <head>:

```html
<!-- dashboard files -->
<script src="js/Dashboard/Dashboard.js"></script>
```

Let's also add a couple extra CSS classes to help on the layout. Add the following to your /css/main.css file:

```css
#dashboard{
  overflow: auto;
  height: calc(100vh - 100px);
}

.transition-width {
  transition: width 1s ease-in-out;
}

.dashboardPanel {
  width: 100%;
  padding: 3%;
  display: block;
}
```

Preparing our data.

The Viewer contains a lot of data from the model, but we need to filter and adjust to our dashboard. The following class will help on that.

There are several ways to organize data, like **Arrays**, for this sample let's use JavaScript object (as a hash table). In essence it would be something like:

```javascript
var data = {};
data['key'] = someValue;
```

But we have multiple levels, like:

```javascript
var data = {};
data['key'] = {};
data['key']['subkey'] = someValue;
```

Using that approach, let's store the property name, property value and an array of dbIds with that value. For instance:

```javascript
data['Category']['Walls'] = [123, 456, 789];
```

The following code will prepare our data.

Let's reuse the **getAllLeafComponents** method (from the Viewer Extension tutorial) to find all the visible dbIds on the model, then use getProperties to get the information.

That is raw data. Under /js/dashboard/ folder, create a new **DashboardPanel.js** with the following content:

```javascript
// Dashboard panel base
class DashboardPanel {
    load(parentDivId, divId, viewer) {
        this.divId = divId;
        this.viewer = viewer;
        $('#' + parentDivId).append('<div id="' + divId + '" class="dashboardPanel"></div>');
    }
}

// Dashboard panels for charts
class DashboardPanelChart extends DashboardPanel {
    load(parentDivId, divId, viewer, modelData) {
        if (!modelData.hasProperty(this.propertyToUse)){
            alert('This model does not contain a ' + this.propertyToUse +' property for the '
            console.log('These are the properties available on this model: ');
            console.log(Object.keys(modelData._modelData));
            return false;
        }
        divId = this.propertyToUse.replace(/[^A-Za-z0-9]/gi, '') + divId; // div name = prope
        super.load(parentDivId, divId, viewer);
        this.canvasId = divId + 'Canvas';
        $('#' + divId).append('<canvas id="' + this.canvasId + '" width="400" height="400"></
        this.modelData = modelData;
        return true;
    }

    generateColors(count) {
        var background = []; var borders = [];
        for (var i = 0; i < count; i++) {
            var r = Math.round(Math.random() * 255); var g = Math.round(Math.random() * 255);
            background.push('rgba(' + r + ', ' + g + ', ' + b + ', 0.2)');
            borders.push('rgba(' + r + ', ' + g + ', ' + b + ', 0.2)');
        }
        return { background: background, borders: borders };
    }
}
```

```
// Model data in format for charts
class ModelData {
    constructor(viewer) {
        this._modelData = {};
        this._viewer = viewer;
    }

    init(callback) {
        var _this = this;

        _this.getAllLeafComponents(function (dbIds) {
            var count = dbIds.length;
            dbIds.forEach(function (dbId) {
                viewer.getProperties(dbId, function (props) {
                    props.properties.forEach(function (prop) {
                        if (!isNaN(prop.displayValue)) return; // let's not categorize proper

                        // some adjustments for revit:
                        prop.displayValue = prop.displayValue.replace('Revit ', ''); // remov
                        if (prop.displayValue.indexOf('<') == 0) return; // skip categories t

                        // ok, now let's organize the data into this hash table
                        if (_this._modelData[prop.displayName] == null) _this._modelData[prop
                        if (_this._modelData[prop.displayName][prop.displayValue] == null) _t
                        _this._modelData[prop.displayName][prop.displayValue].push(dbId);
                    })
                    if ((--count) == 0) callback();
                });
            })
        })
    }

    getAllLeafComponents(callback) {
        // from https://learnforge.autodesk.io/#/viewer/extensions/panel?id=enumerate-leaf-no
        viewer.getObjectTree(function (tree) {
            var leaves = [];
            tree.enumNodeChildren(tree.getRootId(), function (dbId) {
                if (tree.getChildCount(dbId) === 0) {
                    leaves.push(dbId);
                }
            }, true);
            callback(leaves);
        });
    }

    hasProperty(propertyName){
        return (this._modelData[propertyName] !== undefined);
    }

    getLabels(propertyName) {
        return Object.keys(this._modelData[propertyName]);
    }

    getCountInstances(propertyName) {
        return Object.keys(this._modelData[propertyName]).map(key => this._modelData[property
    }

    getIds(propertyName, propertyValue) {
        return this._modelData[propertyName][propertyValue];
    }
}
```

At the **index.html** add a <script> for this new file. This should go inside the <head> and after the **Dashboard.js**:

```html
<script src="js/Dashboard/DashboardPanel.js"></script>
```

Add charts

There are many libraries to create charts, for this sample let's use **Chart.js**, very simple yet nice to use and with great visual.

At the **index.html** add the <script> and <link> stylesheet below for the **Chart.js CDN** libraries reference. This should go inside the <head>

```html
<!--Chart JS  packages-->
<script src="//cdnjs.cloudflare.com/ajax/libs/Chart.js/2.8.0/Chart.min.js" ></script>
<link rel="stylesheet" href="//cdnjs.cloudflare.com/ajax/libs/Chart.js/2.8.0/Chart.min.css" />
```

Under /js/dashboard/ folder create a new **PanelBarChart.js** file with the following content:

```javascript
class BarChart extends DashboardPanelChart {
    constructor(property) {
        super();
        this.propertyToUse = property;
    }

    load(parentDivId, viewer, modelData) {
        if (!super.load(parentDivId, this.constructor.name, viewer, modelData)) return;
        this.drawChart();
    }

    drawChart() {
        var _this = this; // need this for the onClick event

        var ctx = document.getElementById(this.canvasId).getContext('2d');
        var colors = this.generateColors(this.modelData.getLabels(this.propertyToUse).length)

        new Chart(ctx, {
            type: 'bar',
            data: {
                labels: this.modelData.getLabels(this.propertyToUse),
                datasets: [{
                    data: this.modelData.getCountInstances(this.propertyToUse),
                    backgroundColor: colors.background,
                    borderColor: colors.borders,
                    borderWidth: 1
                }]
            },
            options: {
                scales: {
                    yAxes: [{
                        ticks: {
                            beginAtZero: true
                        }
                    }]
                },
                legend: {
                    display: false
                },
                'onClick': function (evt, item) {
                    _this.viewer.isolate(_this.modelData.getIds(_this.propertyToUse, item[0]..
                }
            }
        });
    }
}
```

Under /js/dashboard/ folder create a new **PanelPieChart.js** file with the following content:

```javascript
class PieChart extends DashboardPanelChart {
    constructor(property) {
        super();
        this.propertyToUse = property;
    }

    load(parentDivId, viewer, modelData) {
        if (!super.load(parentDivId, this.constructor.name, viewer, modelData)) return;
        this.drawChart();
    }

    drawChart() {
        var _this = this; // need this for the onClick event

        var ctx = document.getElementById(this.canvasId);
        var colors = this.generateColors(this.modelData.getLabels(this.propertyToUse).length)

        new Chart(ctx, {
            type: 'doughnut',
            data: {
                labels: this.modelData.getLabels(this.propertyToUse),
                datasets: [{
                    data: this.modelData.getCountInstances(this.propertyToUse),
                    backgroundColor: colors.background,
                    borderColor: colors.borders,
                    borderWidth: 1
                }]
            },
            options: {
                legend: {
                    display: true
                },
                'onClick': function (evt, item) {
                    _this.viewer.isolate(_this.modelData.getIds(_this.propertyToUse, item[0]..
                }
            }
        });
    }
}
```

At the **index.html** add 2 <script> for these new files. This should go inside the <head> and after the **DashboardPanel.js**:

```html
<script src="js/Dashboard/PanelBarChart.js"></script>
<script src="js/Dashboard/PanelPieChart.js"></script>
```

Your dashboard is now ready to run!

