CS463313L

90 minutes to Build an Autodesk Forge Application for Project Management

Phuc Le Hieu Hong (Ken)

Technical Specialist, Autodesk

Vo Le Uy

BIM Developer, Cofico

Learning Objectives

- Identify problems in project schedule management and how the Autodesk Forge platform can help overcome these challenges
- Learn how to use the Forge APIs to upload, convert and display your models in a web browser
- Learn how to use the Forge APIs to access and customize your cloud data
- Learn how to use the Forge tool to build a project schedule management application of your own

Description

Project scheduling is an important factor that needs to be strictly controlled to ensure the success of a project. Therefore, contractors need a simple, useful application that can also be accessed online in order to perform project management tasks with BIM models during the construction process. In this class, you will learn how to build a Forge application to visually manage project progress by utilizing BIM models. Forge platform is Cloud-based developer tools from Autodesk. We will introduce Forge APIs by walking through some sample code demonstrations to help you gain access to your BIM data. Forge APIs also provide you the way to manipulate and update more information in BIM models on the cloud. If you are interested in learning Forge, a powerful cloud application, to innovate the way you work and make the best use of your data then this class is for you.

Speakers

Phuc Le is a Digital Consultant, BIM Advisor, BIM Application Expert & Forge Developer.

He currently serves as a Technical Specialist at Autodesk Asean, supporting firms and organizations in the AEC sector to successfully implement Building Information Modelling, Cloud Collaboration, Computational Design, and Generative Design.

Uy Vo Le has a degree in civil engineering but he has been working as a BIM Developer for more than 4 years at Woh Hup, Singapore, and now at Cofico, Vietnam.

In line with his career objective "Bring as many benefits of IT as possible to Construction Management", his works focus on developing Revit add-ins and Forge application.

Table of contents

Autodesk Forge Overview

1. What is Forge

Forge is a cloud-based developer platform from Autodesk.

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

Forge is comprised of the technology and infrastructure the platform it offers, developer/tech resources, and community.

2. Forge APIs

At the time of writing, the Forge platform consists of the following APIs:

- **BIM 360 API**: Build apps and custom integrations for the construction industry
- **Data Management**: Access and manage files and data in Autodesk cloud storage
- **Design Automation**: Run automation scripts on your design files
- **Model Derivative**: Extract data and convert file format of 2D or 3D models
- **Reality Capture**: Create 3D models, orthophotos, and laser scans with photos
- **Token Flex**: Token Flex customers: Access your token usage data
- **Viewer**: View and navigate through complex 2D and 3D models
- **Webhooks**: Receive event notifications from Autodesk Web Services

3. What you can do with Forge

From artificial intelligence to virtual reality, from smart manufacturing to the connected job site—the Forge community of customers and partners are using Forge to innovate in their industries.

Please visit https://forge.autodesk.com/customers for more.

4. Forge application for Project Management

4.1. Next challenges in Project Management

Along with BIM models being widely applied in the construction industry today and the application of Common Data Environment like BIM 360, we have a unified platform connecting your project teams and data in real-time, from design through construction, supporting informed decision-making and leading to more predictable and profitable outcomes.

Till now, the need of customization faces the following challenges (Customization is commonplace.)

- *Customers* who want to improve their project management workflows or create new ones that both drive internal digital innovation and improvements.
- *App DEV Partners* who want to develop new apps to fill in project management workflows.
- *System Integrators* who are looking for a platform to provide software development and consulting services to help companies accelerate their business productivity.

One of the customization needs for project management is scheduling. Time is an important factor that needs to be strictly controlled to ensure the success of a project. Therefore, owners, project managers and contractors need a simple, useful application that can also be accessed online in order to perform project management tasks in sync with BIM models during the construction process.

4.2. Why Forge

Forge can overcome these above challenges. Forge is our end-to-end solution.

Each workflow is slightly different. Rather than having customers with design data bend their operations to meet off-the-shelf solutions, Forge allows them to customize solutions to match their operations.

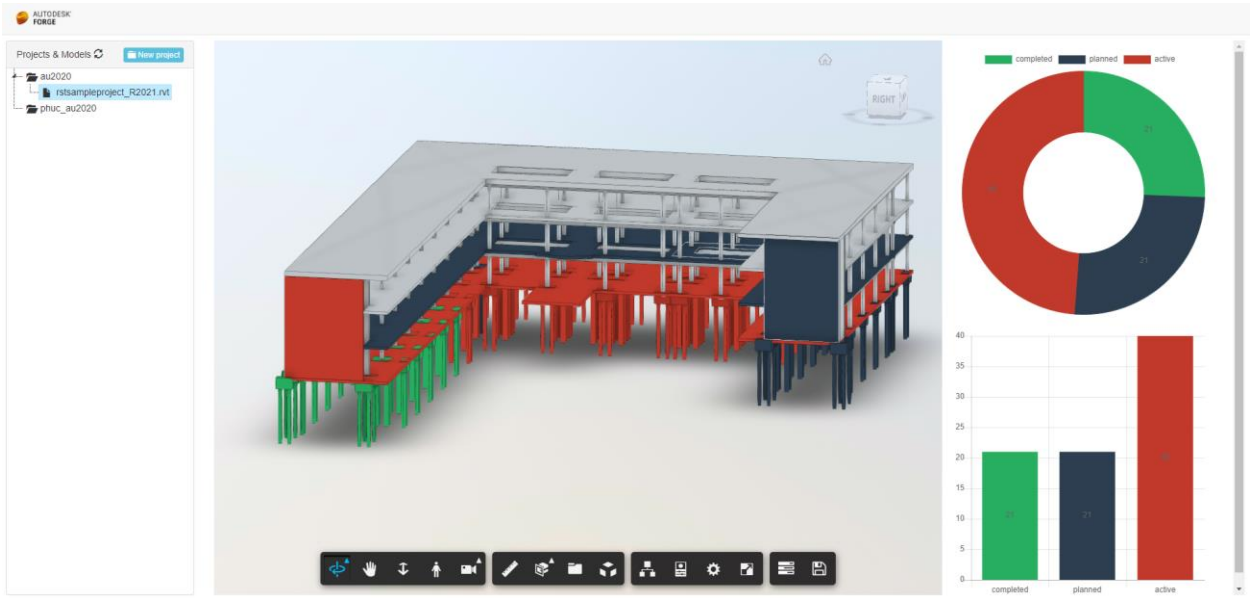
The Data Management API lets users publish their BIM models. The Model Derivative API enables users to translate these models for web viewing on any device, anywhere. Forge APIs also provide users the way to manipulate and update more information in BIM models on the cloud. The entire project status and schedule data are incorporated into each element in BIM models via Forge application. This is the single source of truth for project management

Build your Forge application for project management

1. Overview

1.1. Description

This lab will cover how to build a Forge application to visually manage project progress by utilizing BIM models. To be specific, we will develop a simple web page that allows project teams (such as site engineers, supervisors, or project managers) to upload and view 3D models and their data on a browser. In the web view, we can easily update elements' status in real-time to keep good track of the project's progress at the same time.



1.2. Who is this for

If you are interested in learning Forge, a powerful cloud application, to innovate the way you work and make the best use of your data then this class is for you.

The content in this lab is intended for anyone who has a basic knowledge of web development.

In case you are a complete novice at web development or coding, do not worry because there is no need for you to fully understand all the sample codes provided. You just have to go through the lesson and that would subsequently help you gain a better view of Forge and build up more creative ideas to make anything that suits your need.

1.3. What are the APIs used

This tutorial uses 4 APIs:

- **Authentication** (Oauth): generate tokens based on the Oauth 2.0 specification, which are used for authorizing requests sent to Forge APIs
- **Model Derivative**: Derive outputs for viewing, extracts object data, and converts design files to industry standard formats
- **Data Management**: Access and manage files and data in Autodesk cloud storage
- **Viewer**: Render 3D and 2D model data within a browser

1.4. What you will get from this lab

At the end of this tutorial, you will know how to:

- Run a simple web server
- Serve a web page
- Authenticate your developer identity
- Create a bucket
- Save a supported file to a bucket
- Read the uploaded file from a bucket
- Translate the file to the SVF format
- Show a 3D file in a web viewer
- Add custom properties to models' objects.
- Save added data
- Getting current data of models
- Retrieve added custom data for displaying on dashboard
- Add dashboard: bar chart and pie chart

Note that in this lab, we will be setting up a Web server using Express for Node.js. It is purely for serving a web page for our tutorial, not a requirement for use in any particular Forge API.

1.5. What is required

- A Forge account (Register now for a free trial)

(Trial duration**:** free trials last 90 days or until your 100 Cloud Credits are consumed - whichever happens first).

- A text editor of your choice. (For example, Brackets or Visual Studio Code are good choices – this class uses Visual Studio Code)
- A basic knowledge of:
    - HTML and CSS
    - JavaScript (because Node.js is based on JavaScript)
    - Command-line programs
    - Node.js Command Line (for Windows users)
    - Terminal (for Mac/Linux/Unix users)
    - Express Basic Routing
    - Module FS
    - jQuery

2. Setup

Let's set up your machine before we begin the walkthrough properly.

2.1. Getting test environment

Let's install some software on your machine.

a. Node.js

Install **Node.js** engine to run your code. Install **NPM** dependency manager to install packages.

Download and install the latest version of Node.js (version 8 or higher) from this link: https://nodejs.org/en/download/

npm is installed with Node.js - which means that when you download Node.js, you automatically get npm installed on your computer.

Once installed, open a terminal window (or on Windows, a Node.js command prompt window). The terminal will be used to enter Node.js commands.

Check that you have node and npm installed:

To check if you have Node.js installed, run this command in your terminal:

```
node -v
```

To confirm that you have npm installed you can run this command in your terminal:

```
npm -v
```

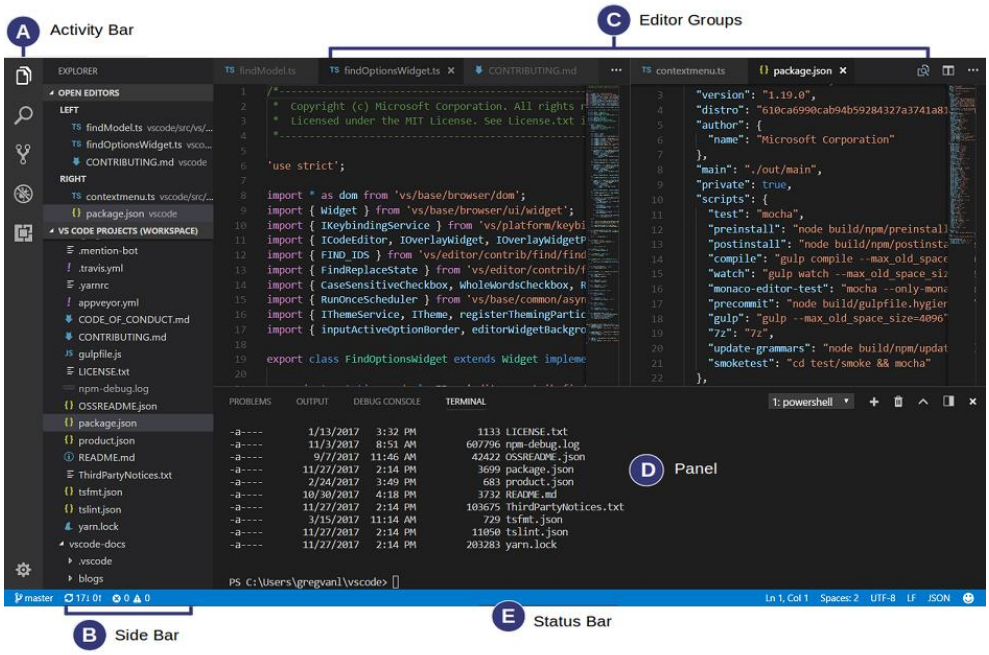
b. Visual Studio Code

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity)

Download and install the latest version of Visual Studio Code from this link:

https://code.visualstudio.com/

The interface of Visual Studio Code looks like:



Please visit https://code.visualstudio.com/docs/getstarted/introvideos for introductory videos.

2.2. Getting Forge Client Id and Client Secret

a. Creating a Forge account

You must have an Autodesk Account to get a Forge account.

If you don't have Autodesk Account, go to https://forge.autodesk.com/ and click **Create Account.**

If you already have an Autodesk Account, sign in with your Autodesk Account. A Forge account is automatically created when you sign in to forge.autodesk.com.

b. Registering an app

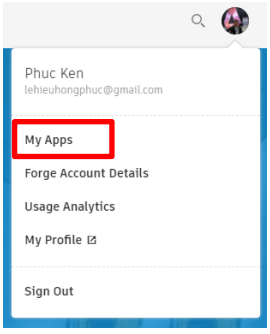
Registering an app grants it permission it to use Forge services. At registration you are given a Client ID to use in the app. The Client ID uniquely identifies your app. This means that:

- The Client ID enables your app to access Forge services across all platforms.
- Deleting an app from the My Apps page will prevent your from accessing Forge Services.
- Forge services utilized by your app will be charged to your account.
- Deleting an app from the My Apps merely removes it from that page. History and billing records remain.

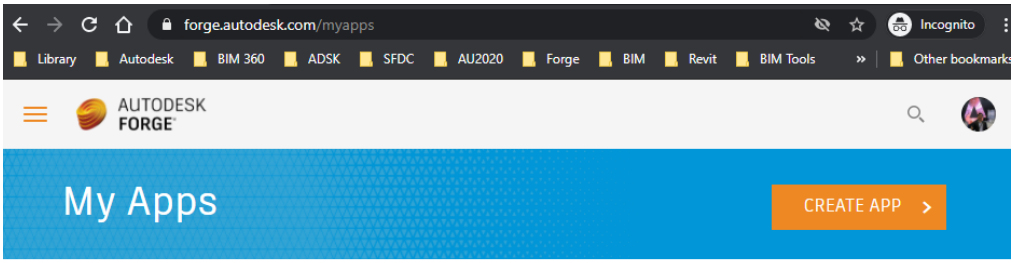
To register an app:

a. Sign in to Forge.

b. In the Profile drop-down, click **My Apps** or click **Go to My Apps**



c. Within the My Apps page, click **Create App**

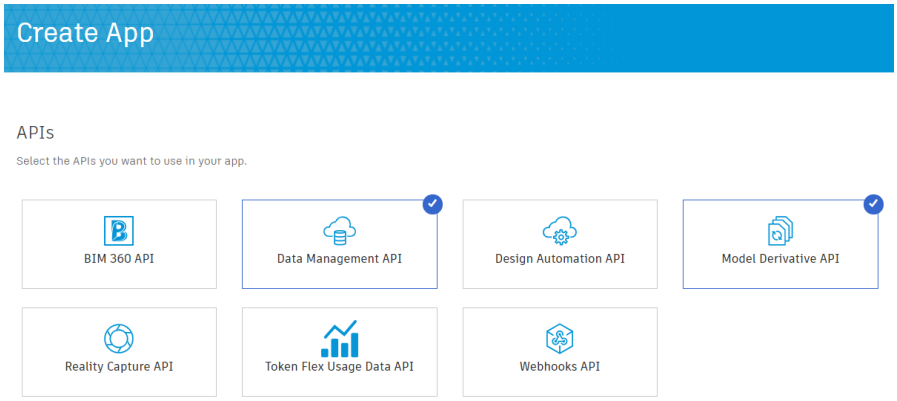


d. Select all the services you want to use for your app.

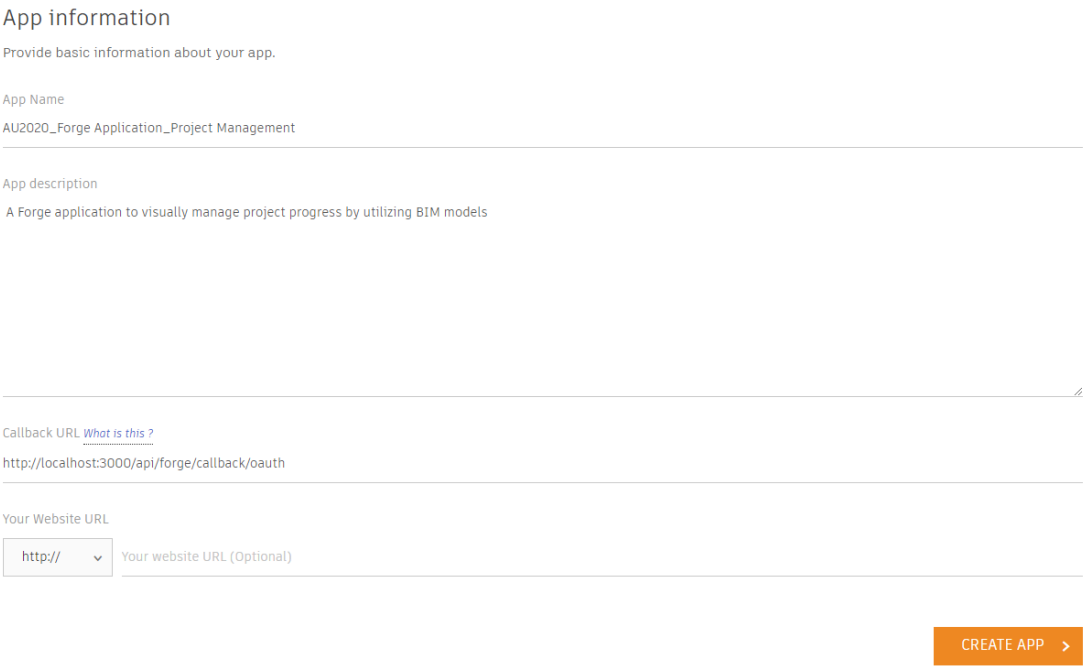
In this class, please select the "**Data Management API**" and "**Model Derivative API**".

Note that all APIs are typically pre-selected, so you may have to unselect everything except Data Management and Model Derivative.

The Authentication and Viewer API are default services, so you will call these two APIs when coding.



e. Fill in the mandatory fields in the App Information section.



Name the app “AU2020_Forge Application_Project Management” or you can add any name as you want.

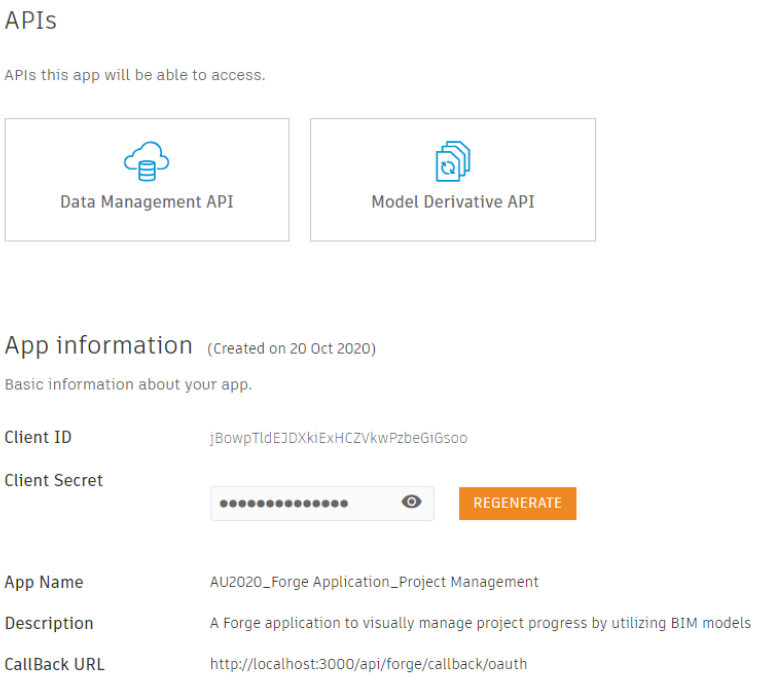
For “Callback URL”, enter **http://localhost:3000/api/forge/callback/oauth.** Although we will not use the Callback URL in this walkthrough, we need to specify a valid URL as the Callback URL box cannot be left empty.

> Note that a callback URL is required for 3-legged OAuth authentication.
>
> If you do not have a callback URL as yet, or do not require 3-legged authorization, you can specify an arbitrary URL to complete this field.

Then, click **CREATE APP** to complete the operation.

c. Getting Client ID and Secret:

Now that you have created a Forge app, you are assigned a Client ID and Client Secret, which are essentially the credentials issued to your app, granting it access to Forge services.



- In the My Apps page, click the newly created app.
- Within the app details page, scroll down to the App information section.

  Here you will see both the Client ID and Client Secret.

  *The **Client Secret** is masked by default. Click the eye icon to reveal its value.*

- Make a note of your **Client ID** and **Client Secret**. They will be used to generate a token, which will be appended to every request your app makes. The token lets Forge authenticate each request and associate them with your account.

2.3. Getting the datasets

You can download datasets included the code sample and the sample Revit file in the datasets on the AU class page or at the following link:

https://drive.google.com/open?id=1m-aiOgCi79e0Wm7SM9zwy7UQLCPaC727

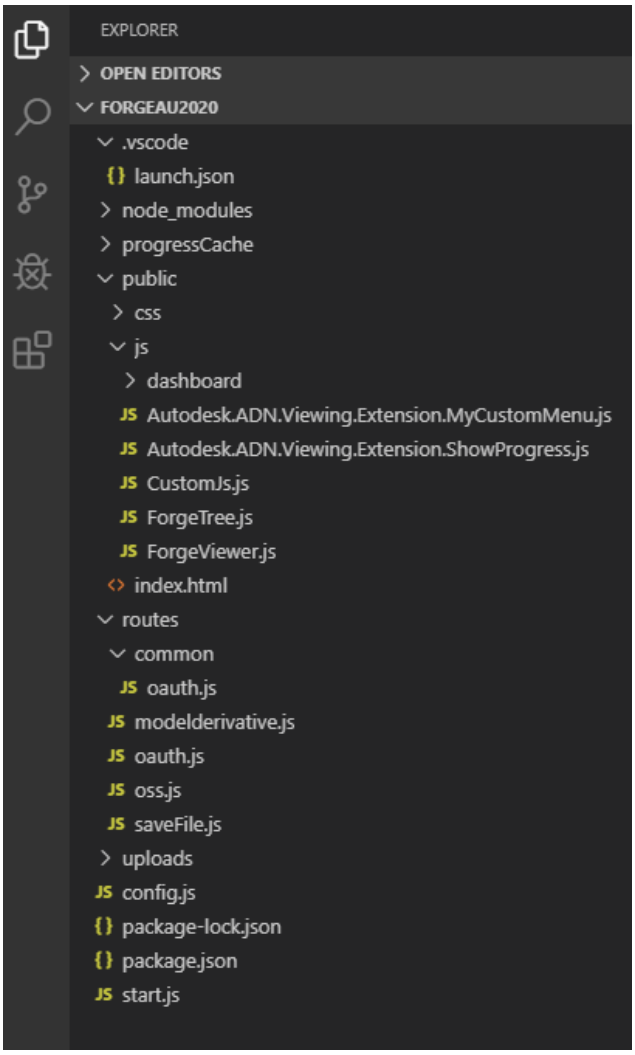
3. Understanding the code

Let's take a close look at the code and try to understand the underlying logic.

We will learn the Step-by-step tutorial for:

- View your models: Upload, translate and display your models in a web viewer
- Modify cloud data: Add and update custom properties to models' objects
- Dashboard: Retrieve data and create a dashboard with some charts

First, Open **Visual Studio Code**, then go to menu **File** and select **Open Folder** (Windows) or **Open** (MacOS) and select the "ForgeAU2020" folder in **datasets**, then click **Select Folder** button. The code sample will open with the following structure:



3.1. View your models

a. Web Server

We will cover the essential of setting up a basic web server. If you want to find out more, please visit https://expressjs.com/

Let's open the file **start.js**. This file starts an express server, serves static files (e.g. html), and routes API requests.

```
1.  const path = require('path');
2.  const express = require('express');
3.
4.  const PORT = process.env.PORT || 3000;
5.  const config = require('./config');
6.  if (config.credentials.client_id == null || config.credentials.client_secret == null) {
7.      console.error('Missing FORGE_CLIENT_ID or FORGE_CLIENT_SECRET env. variables.');
8.      return;
9.  }
10.
11. let app = express();
12. app.use(express.static(path.join(__dirname, 'public')));
13. app.use(express.json({ limit: '50mb' }));
14. app.use('/api/forge/oauth', require('./routes/oauth'));
15. app.use('/api/forge/oss', require('./routes/oss'));
16. app.use('/api/forge/saveFile', require('./routes/saveFile'));
17. app.use('/api/forge/modelderivative', require('./routes/modelderivative'));
18. app.use((err, req, res, next) => {
19.     console.error(err);
20.     res.status(err.statusCode).json(err);
21. });
22. app.listen(PORT, () => { console.log(`Server listening on port ${PORT}`); });
```

We configure our Express web server to accept the JSON file format and set up our static files to be saved in the folder named ‘public’

@**start.js** Line 11 to 13:

```
11. let app = express();
12. app.use(express.static(path.join(__dirname, 'public')));
13. app.use(express.json({ limit: '50mb' }));
```

Finally, we set up the server to listen to port 3000. Because of this, when you access our app, you must append :3000 to the URL.

For example, to access localhost you must specify http://localhost:3000, instead of just http://localhost.

@start.js Line 4 to 9:

```
4.  const PORT = process.env.PORT || 3000;
5.  const config = require('./config');
6.  if (config.credentials.client_id == null || config.credentials.client_secret == null) {
7.      console.error('Missing FORGE_CLIENT_ID or FORGE_CLIENT_SECRET env. variables.');
8.      return;
9.  }
```

Next, let's look at the file **launch.json** in /.vscode folder. This file indicates to **Visual Studio Code** how we should run our project.

> *Noted: You can create this file in new project by Go to menu Debug >> Add Configuration... and, in the Select Environment window that appears on the top, choose Node.js. In the /.vscode/launch.json file that is created, enter your code inside.*

Open the file **launch.json** in /.vscode folder, the sample code is shown as below:

```
1.  {
2.      // Use IntelliSense to learn about possible attributes.
3.      // Hover to view descriptions of existing attributes.
4.      // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
5.      "version": "0.2.0",
6.      "configurations": [
7.          {
8.              "type": "node",
9.              "request": "launch",
10.             "name": "Launch Program",
11.             "skipFiles": [
12.                 "<node_internals>/**"
13.             ],
14.             "program": "${workspaceFolder}\\start.js",
15.             "env": {
16.                 //"FORGE_CLIENT_ID": "Your Client ID",
17.                 "FORGE_CLIENT_ID": "dGDFJ4cdNqbtaPTzVpVSDrpQLg3vmzAQ",
18.                 //"FORGE_CLIENT_SECRET": "Your Client Secret",
19.                 "FORGE_CLIENT_SECRET": "L5b5888cbeea74b3",
20.                 "FORGE_CALLBACK_URL": "http://localhost:3000/api/forge/callback/oauth"
21.             }
22.         },
23.         {
24.           "type": "chrome",
25.           "request": "launch",
26.           "name": "Chrome",
27.           "url": "http://localhost:3000",
28.           "webRoot": "${workspaceRoot}/public"
29.         }
30.     ]
31. }
```

Note you need to **enter your** Forge Client ID & Secret **of your Forge app** (when you create new Forge app - 2.2 section) at the indicated space in line 16 and line 18 and use “//” in line 17 and line 19 to ignore the sample ID & Secret.

The updated **/.vscode/launch.json** Line 16-19 should show as below:

```
16.                 "FORGE_CLIENT_ID": "Update your Forge Client ID",
17.                 //"FORGE_CLIENT_ID": "dGDFJ4cdNqbtaPTzVpVSDrpQLg3vmzAQ",
18.                 "FORGE_CLIENT_SECRET": "Update Your Client Secret",
19.                 //"FORGE_CLIENT_SECRET": "L5b5888cbeea74b3",
```

In the root folder, open the file **config.js**:

```
1.  // Autodesk Forge configuration
2.  module.exports = {
3.      // Set environment variables or hard-code here
4.      credentials: {
5.          client_id: process.env.FORGE_CLIENT_ID,
6.          client_secret: process.env.FORGE_CLIENT_SECRET,
7.          callback_url: process.env.FORGE_CALLBACK_URL
8.      },
9.      scopes: {
10.         // Required scopes for the server-side application
11.         internal: ['bucket:create', 'bucket:read', 'data:read', 'data:create', 'data:wr
    ite'],
12.         // Required scope for the client-side viewer
13.         public: ['viewables:read']
14.     }
15. };
```

We are defining our ENV variables here. At the time of running our Express server, the values of these variables will be used to connect to different Autodesk Forge services we need.

Last, we can see that there are 2 scope definitions. The internal scope gives our access token the right permission for the use of the different services of the Forge Web Services (server-side). This tutorial is dedicated to the use of the Viewer, we will only need the "viewables:read" scope for public.

b. Authenticate

Forge supports two types of authentication:

- Two-Legged Authentication
- Three-Legged Authentication

In this application, we will be using **Two-Legged Authentication**.

In short, Two-Legged authentication is simplest and usually the default in many API libraries.

This kind of flow is called *two-legged* because your app and the Forge Platform are the two legs.

Refer to this API Basics for more information about those 2 types of authentications.

For a basic *OAuth* implementation we need 2 files.

Open the **/routes/oauth.js** file. This file takes care of creating an express router for OAuth-related endpoints.

```
1.  const express = require('express');
2.
3.  const { getPublicToken } = require('./common/oauth');
4.
5.  let router = express.Router();
6.
7.  // GET /api/forge/oauth/token - generates a public access token (required by the Forge
    viewer).
8.  router.get('/token', async (req, res, next) => {
9.      try {
10.         const token = await getPublicToken();
11.         res.json({
12.             access_token: token.access_token,
13.             expires_in: token.expires_in
14.         });
15.     } catch(err) {
16.         next(err);
17.     }
18. });
19.
20. module.exports = router;
```

And the second file **/routes/common/oauth.js** . This file will actually request the access token from Forge. This will be reused in other parts of this tutorial.

```
1.  const { AuthClientTwoLegged } = require('forge-apis');
2.
3.  const config = require('../../config');
4.
5.  /**
6.   * Initializes a Forge client for 2-legged authentication.
7.   * @param {string[]} scopes List of resource access scopes.
8.   * @returns {AuthClientTwoLegged} 2-legged authentication client.
9.   */
10. function getClient(scopes) {
11.     const { client_id, client_secret } = config.credentials;
```

```
12.     return new AuthClientTwoLegged(client_id, client_secret, scopes || config.scopes.in
    ternal);
13. }
14.
15. let cache = {};
16. async function getToken(scopes) {
17.     const key = scopes.join('+');
18.     if (cache[key]) {
19.         return cache[key];
20.     }
21.     const client = getClient(scopes);
22.     let credentials = await client.authenticate();
23.     cache[key] = credentials;
24.     setTimeout(() => { delete cache[key]; }, credentials.expires_in * 1000);
25.     return credentials;
26. }
27.
28. /**
29.  * Retrieves a 2-legged authentication token for preconfigured public scopes.
30.  * @returns Token object: { "access_token": "...", "expires_at": "...", "expires_in": "
    ...", "token_type": "..." }.
31.  */
32. async function getPublicToken() {
33.     return getToken(config.scopes.public);
34. }
35.
36. /**
37.  * Retrieves a 2-legged authentication token for preconfigured internal scopes.
38.  * @returns Token object: { "access_token": "...", "expires_at": "...", "expires_in": "
    ...", "token_type": "..." }.
39.  */
40. async function getInternalToken() {
41.     return getToken(config.scopes.internal);
42. }
43.
44. module.exports = {
45.     getClient,
46.     getPublicToken,
47.     getInternalToken
48. };
```

c. Upload file to OSS

In Data Management or Forge OSS (Object Storage Service), files are stored in containers known as buckets. Apart from giving your app the ability to download data from the broader Forge ecosystem, it also provides the functionality to manage your app's own buckets and objects (including creation, listing, deleting, uploading, and downloading).

In this section we need 3 features:

- Creating buckets
- Listing buckets & objects (files)
- Uploading objects (files)

Open the **/routes/oos.js**:

```
1.  const fs = require('fs');
2.  const express = require('express');
3.  const multer  = require('multer');
4.  const { BucketsApi, ObjectsApi, PostBucketsPayload } = require('forge-apis');
5.
6.  const { getClient, getInternalToken } = require('./common/oauth');
7.  const config = require('../config');
8.
9.  let router = express.Router();
10.
11. // Middleware for obtaining a token for each request.
12. router.use(async (req, res, next) => {
13.     const token = await getInternalToken();
14.     req.oauth_token = token;
15.     req.oauth_client = getClient();
16.     next();
17. });
18.
19. // GET /api/forge/oss/buckets - expects a query param 'id'; if the param is '#' or empt
    y,
20. // returns a JSON with list of buckets, otherwise returns a JSON with list of objects i
    n bucket with given name.
21. router.get('/buckets', async (req, res, next) => {
22.     const bucket_name = req.query.id;
23.     if (!bucket_name || bucket_name === '#') {
24.         try {
25.             // Retrieve buckets from Forge using the [BucketsApi](https://github.com/Au
    todesk-Forge/forge-api-nodejs-client/blob/master/docs/BucketsApi.md#getBuckets)
26.             const buckets = await new BucketsApi().getBuckets({ limit: 64 }, req.oauth_
    client, req.oauth_token);
27.             res.json(buckets.body.items.map((bucket) => {
28.                 return {
29.                     id: bucket.bucketKey,
30.                     // Remove bucket key prefix that was added during bucket creation
31.                     text: bucket.bucketKey.replace(config.credentials.client_id.toLower
    Case() + '-', ''),
32.                     type: 'bucket',
33.                     children: true
34.                 };
35.             }));
36.         } catch(err) {
37.             next(err);
38.         }
39.     } else {
40.         try {
41.             // Retrieve objects from Forge using the [ObjectsApi](https://github.com/Au
    todesk-Forge/forge-api-nodejs-client/blob/master/docs/ObjectsApi.md#getObjects)
42.             const objects = await new ObjectsApi().getObjects(bucket_name, {}, req.oaut
    h_client, req.oauth_token);
43.             res.json(objects.body.items.map((object) => {
44.                 return {
45.                     id: Buffer.from(object.objectId).toString('base64'),
46.                     text: object.objectKey,
```

```
47.                     type: 'object',
48.                     children: false
49.                 };
50.             }));
51.         } catch(err) {
52.             next(err);
53.         }
54.     }
55. });
56.
57. // POST /api/forge/oss/buckets - creates a new bucket.
58. // Request body must be a valid JSON in the form of { "bucketKey": "<new_bucket_name>"
    }.
59. router.post('/buckets', async (req, res, next) => {
60.     let payload = new PostBucketsPayload();
61.     payload.bucketKey = config.credentials.client_id.toLowerCase() + '-
    ' + req.body.bucketKey;
62.     payload.policyKey = 'transient'; // expires in 24h
63.     try {
64.         // Create a bucket using [BucketsApi](https://github.com/Autodesk-Forge/forge-
    api-nodejs-client/blob/master/docs/BucketsApi.md#createBucket).
65.         await new BucketsApi().createBucket(payload, {}, req.oauth_client, req.oauth_to
    ken);
66.         res.status(200).end();
67.     } catch(err) {
68.         next(err);
69.     }
70. });
71.
72. // POST /api/forge/oss/objects - uploads new object to given bucket.
73. // Request body must be structured as 'form-data' dictionary
74. // with the uploaded file under "fileToUpload" key, and the bucket name under "bucketKe
    y".
75. router.post('/objects', multer({ dest: 'uploads/' }).single('fileToUpload'), async (req
    , res, next) => {
76.     fs.readFile(req.file.path, async (err, data) => {
77.         if (err) {
78.             next(err);
79.         }
80.         try {
81.             // Upload an object to bucket using [ObjectsApi](https://github.com/Autodes
    k-Forge/forge-api-nodejs-client/blob/master/docs/ObjectsApi.md#uploadObject).
82.             await new ObjectsApi().uploadObject(req.body.bucketKey, req.file.originalna
    me, data.length, data, {}, req.oauth_client, req.oauth_token);
83.             res.status(200).end();
84.         } catch(err) {
85.             next(err);
86.         }
87.     });
88. });
89.
90. module.exports = router;
```

In the line 61-62, we define a bucket key and policy key when creating a bucket

```
61. payload.bucketKey = config.credentials.client_id.toLowerCase() + '-
    ' + req.body.bucketKey;
62.     payload.policyKey = 'transient'; // expires in 24h
```

We can set any name as the bucket key. Thereafter, whenever we upload objects to the bucket, we will use the bucket key to access the bucket.

The bucket key must be globally unique across all apps and regions, if not the call will fail.

That is why we have prefixed it with your ID so the bucket key is unique across all buckets on all other accounts.

***Note that bucket keys** must be of the form [-_.a-z0-9]{3,128}*

Each bucket also has a retention policy that determines object retention time, it is the policy key:

- **transient**: Cache-like storage that persists for only 24 hours, ideal for ephemeral objects.
- **temporary**: Storage that persists for 30 days.
- **persistent**: Storage that persists until it's deleted.

For this sample, let's use policy key is **transient**.

Since we plan to support jsTree, our GET **/api/forge/oss/buckets** endpoint needs to handle the id query string parameter, returning all buckets when id is set to #, or returning all objects in a given bucketKey passed as id=bucketKey. The upload endpoint uses the **multer** module to handle file upload. It saves the file on our server (e.g. in /uploads/ folder) so we can later upload it to Forge.

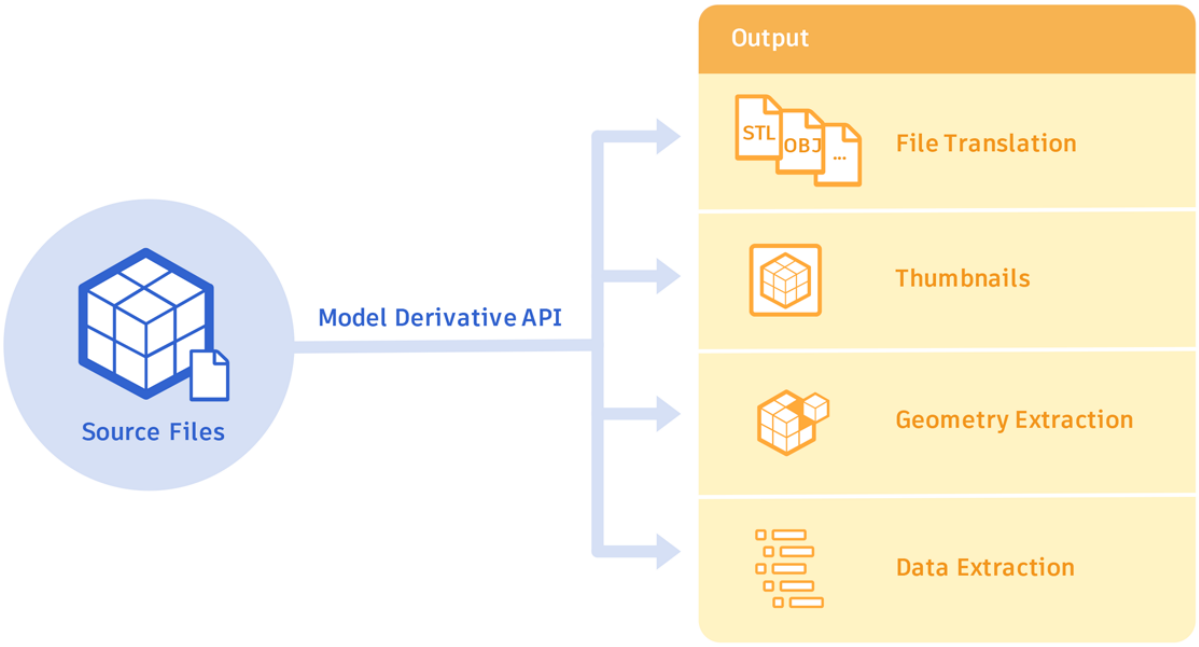
Note how we reuse the authentication helpers from **/routes/common/oauth.js** as a middleware of this router.

d. Translate

Next, we will translate the uploaded file into a format that we can display in the viewer. In this walkthrough, we will translate it into the SVF format using the Model Derivative API.

The Model Derivate API enables you to translate over 70 different types of files into derivatives (output files derived from a source file of another format).

Visit supported translations for a complete list of supported formats.



Let's create a new route named **/api/forge/modelderivative** to help us with the translation.

Open **/routes/modelderivative.js**:

```
1.  const express = require('express');
2.  const {
3.      DerivativesApi,
4.      JobPayload,
5.      JobPayloadInput,
6.      JobPayloadOutput,
7.      JobSvfOutputPayload
8.  } = require('forge-apis');
9.
10. const { getClient, getInternalToken } = require('./common/oauth');
11.
12. let router = express.Router();
13.
14. // Middleware for obtaining a token for each request.
15. router.use(async (req, res, next) => {
16.     const token = await getInternalToken();
17.     req.oauth_token = token;
18.     req.oauth_client = getClient();
19.     next();
20. });
21.
22. // POST /api/forge/modelderivative/jobs - submits a new translation job for given objec
    t URN.
23. // Request body must be a valid JSON in the form of { "objectName": "<translated-
    object-urn>" }.
24. router.post('/jobs', async (req, res, next) => {
25.     let job = new JobPayload();
26.     job.input = new JobPayloadInput();
```

```
27.     job.input.urn = req.body.objectName;
28.     job.output = new JobPayloadOutput([
29.         new JobSvfOutputPayload()
30.     ]);
31.     job.output.formats[0].type = 'svf';
32.     job.output.formats[0].views = ['2d', '3d'];
33.     try {
34.         // Submit a translation job using [DerivativesApi](https://github.com/Autodesk-
    Forge/forge-api-nodejs-client/blob/master/docs/DerivativesApi.md#translate).
35.         await new DerivativesApi().translate(job, {}, req.oauth_client, req.oauth_token
    );
36.         res.status(200).end();
37.     } catch(err) {
38.         next(err);
39.     }
40. });
41.
42. module.exports = router;
```

We then set the output type to **svf** and views to **2d** and **3d** in Line 31-32.

In summary, the jobs endpoint receives the objectName and posts the translation job to translate it into the **svf** format for 2D and 3D views.

e. Viewer

Up until now, all the JavaScript we wrote was within **start.js.**

This script runs on the server.

All the functions and variables are only accessible to the server. Client-side users have no way to see or control the code from their browser.

We call this a **backend** JavaScript.

All the heavy lifting of displaying our SVF file is done by the JavaScript within **index.html**.

Any JavaScript between the *<script>* and *</script>* tags within a HTML file, is referred to as **frontend** JavaScript.

The Autodesk and jQuery scripts we imported earlier are also considered as **Frontend** scripts.

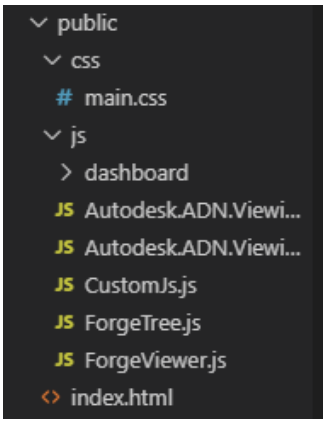
Difference between **Frontend** JavaScript and **Backend** JavaScript:

Frontend JavaScript	**Backend JavaScript**
Referenced from within an HTML file.	Not placed inside or accessed from within an HTML file.
Runs on the end user's web browser and hence is visible to end users and can be changed by them.	Runs on a Node.js server and is not visible to end user on their browser.
Changes to the code only require a page refresh.	Changes to the code require restarting the Node.js server.
The script must not contain any sensitive data.	Can contain sensitive data such as the API key.
Example: index.html	Example: start.js

The Viewer is a client-side library, based on pure HTML5 and JavaScript. But there are a few tips for each server-side implementation:

Our Node.js server is configured to serve files from public folder. Let's organize its content like this:

- public/: .html
- public/js: .js
- public/css: .css



Index.html

This is the entry point of your app.

For this sample we'll use jQuery for DOM manipulation, Bootstrap for styling and jsTree to list buckets & objects. All those libraries are coming from CDN (Content Delivery Network).

And, of course, the Autodesk Forge Viewer libraries: viewer3d.min.js, three.min.js and style.min.css.

The **/public/index.html** file has the following content:

```
<!DOCTYPE html>
<html>
<head>
  <title>View Models - Autodesk Forge</title>
  <meta charset="utf-8" />
  <link rel="shortcut icon" href="https://github.com/Autodesk-Forge/learn.forge.viewmodels/raw/master/img/favicon.ico">
  <!-- Common packages: jQuery, Bootstrap, jsTree -->
  <script src="//cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
  <script src="//cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/3.4.1/js/bootstrap.min.js"></script>
  <script src="//cdnjs.cloudflare.com/ajax/libs/jstree/3.3.7/jstree.min.js"></script>
  <link rel="stylesheet" href="//cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/3.4.1/css/bootstrap.min.css">
  <link rel="stylesheet" href="//cdnjs.cloudflare.com/ajax/libs/jstree/3.3.7/themes/default/style.min.css" />
  <!-- Autodesk Forge Viewer files -->
  <link rel="stylesheet" href="https://developer.api.autodesk.com/modelderivative/v2/viewers/7.*/style.min.css" type="text/css">
  <script src="https://developer.api.autodesk.com/modelderivative/v2/viewers/7.*/viewer3D.min.js"></script>
  <!-- this project files -->
  <link href="css/main.css" rel="stylesheet" />
  <script src="js/ForgeTree.js"></script>
  <script src="js/ForgeViewer.js"></script>
  <script src="js/Autodesk.ADN.Viewing.Extension.MyCustomMenu.js"></script>
  <script src="/js/Autodesk.ADN.Viewing.Extension.ShowProgress.js"></script>

  <script src="js/CustomJs.js"></script>

<!--Chart JS  packages-->
<script src="//cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.3/Chart.min.js" ></script>
<link rel="stylesheet" href="//cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.3/Chart.min.css" />

<!--For show the values on all charts.-->
<script src="https://cdn.jsdelivr.net/npm/chart.js@2.7.3/dist/Chart.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/chartjs-plugin-datalabels@0.7.0"></script>

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

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


</head>
<body>
  <!-- Fixed navbar by Bootstrap: https://getbootstrap.com/examples/navbar-fixed-top/ -->
  <nav class="navbar navbar-default navbar-fixed-top">
    <div class="container-fluid">
      <ul class="nav navbar-nav left">
        <li>
          <a href="http://developer.autodesk.com" target="_blank">
            <img alt="Autodesk Forge" src="//developer.static.autodesk.com/images/logo_forge-2-line.png" height="20">
          </a>
        </li>
      </ul>
    </div>
  </nav>
  <!-- End of navbar -->
  <div class="container-fluid fill">
    <div class="row fill">
      <div class="col-sm-4 fill">
        <div class="panel panel-default fill">
          <div class="panel-heading" data-toggle="tooltip">
            Projects & Models
            <span id="refreshBuckets" class="glyphicon glyphicon-refresh" style="cursor: pointer"></span>
            <button class="btn btn-xs btn-info" style="float: right" id="showFormCreateBucket" data-toggle="modal" data-target="#createBucketModal">
              <span class="glyphicon glyphicon-folder-close"></span> New project
            </button>
          </div>
          <div id="appBuckets">
            tree here
          </div>
        </div>
      </div>
      <div class="col-sm-8 fill">
        <div id="forgeViewer"></div>
      </div>
    </div>
  </div>
  <form id="uploadFile" method='post' enctype="multipart/form-data">
    <input id="hiddenUploadField" type="file" name="theFile" style="visibility:hidden" />
  </form>
  <!-- Modal Create Bucket -->
  <div class="modal fade" id="createBucketModal" tabindex="-1" role="dialog" aria-labelledby="myModalLabel">
    <div class="modal-dialog" role="document">
      <div class="modal-content">
        <div class="modal-header">
          <button type="button" class="close" data-dismiss="modal" aria-label="Cancel">
            <span aria-hidden="true">×</span>
```

```
86.           </button>
87.           <h4 class="modal-title" id="myModalLabel">Create new project</h4>
88.         </div>
89.         <div class="modal-body">
90.           <input type="text" id="newBucketKey" class="form-
    control"> For demonstration purposes, objects (files) are
91.           NOT automatically translated. After you upload, right click on
92.           the object and select "Translate". Bucket keys must be of the form [-_.a-z0-
    9]{3,128}
93.         </div>
94.         <div class="modal-footer">
95.           <button type="button" class="btn btn-default" data-
    dismiss="modal">Cancel</button>
96.           <button type="button" class="btn btn-
    primary" id="createNewBucket">Go ahead, create the project</button>
97.         </div>
98.       </div>
99.     </div>
100.         </div>
101.       </body>
102.       </html>
```

Main.css

CSS is a language that describes the style of HTML documents. Learn more at W3Schools. For this class, we have a main.css under the css folder with the following content:

```
1.  html, body {
2.      min-height: 100%;
3.      height: 100%;
4.    }
5.
6.    .fill {
7.      height: calc(100vh - 100px);
8.    }
9.
10.   body {
11.     padding-top: 60px; /* space for the top nav bar */
12.     margin-right: 30px;
13.   }
14.
15.   #appBuckets {
16.     overflow: auto;
17.     width: 100%;
18.     height: calc(100vh - 150px);
19.   }
20.
21.   #forgeViewer {
22.     width: 100%;
23.   }
24.
25.   #dashboard{
26.     overflow: auto;
27.     height: calc(100vh - 100px);
28.   }
```

```
29.
30.   .transition-width {
31.     transition: width 1s ease-in-out;
32.   }
33.
34.   .dashboardPanel {
35.     width: 100%;
36.     padding: 3%;
37.     display: block;
38.   }
39.
40.   .showProgressIcon {
41.     background-image: url(https://raw.githubusercontent.com/encharm/Font-Awesome-SVG-
    PNG/master/white/png/24/tasks.png);
42.     background-size: 24px;
43.     background-repeat: no-repeat;
44.     background-position: center;
45.   }
46.
47.   .saveProgressIcon {
48.     background-image: url(https://raw.githubusercontent.com/encharm/Font-Awesome-SVG-
    PNG/master/white/png/24/save.png);
49.     background-size: 24px;
50.     background-repeat: no-repeat;
51.     background-position: center;
52.   }
```

ForgeTree.js

This file will handle the tree view that lists all your buckets.

```
1.  $(document).ready(function () {
2.      prepareAppBucketTree();
3.      $('#refreshBuckets').click(function () {
4.        $('#appBuckets').jstree(true).refresh();
5.      });
6.
7.      $('#createNewBucket').click(function () {
8.        createNewBucket();
9.      });
10.
11.     $('#createBucketModal').on('shown.bs.modal', function () {
12.       $("#newBucketKey").focus();
13.     })
14.
15.     $('#hiddenUploadField').change(function () {
16.       var node = $('#appBuckets').jstree(true).get_selected(true)[0];
17.       var _this = this;
18.       if (_this.files.length == 0) return;
19.       var file = _this.files[0];
20.       switch (node.type) {
21.         case 'bucket':
22.           var formData = new FormData();
23.           formData.append('fileToUpload', file);
24.           formData.append('bucketKey', node.id);
25.
```

```
26.           $.ajax({
27.             url: '/api/forge/oss/objects',
28.             data: formData,
29.             processData: false,
30.             contentType: false,
31.             type: 'POST',
32.             success: function (data) {
33.               $('#appBuckets').jstree(true).refresh_node(node);
34.               _this.value = '';
35.             }
36.           });
37.           break;
38.       }
39.     });
40.   });
41.
42.   function createNewBucket() {
43.     var bucketKey = $('#newBucketKey').val();
44.     jQuery.post({
45.       url: '/api/forge/oss/buckets',
46.       contentType: 'application/json',
47.       data: JSON.stringify({ 'bucketKey': bucketKey }),
48.       success: function (res) {
49.         $('#appBuckets').jstree(true).refresh();
50.         $('#createBucketModal').modal('toggle');
51.       },
52.       error: function (err) {
53.         if (err.status == 409)
54.           alert('Bucket already exists - 409: Duplicated')
55.         console.log(err);
56.       }
57.     });
58.   }
59.
60.   function prepareAppBucketTree() {
61.     $('#appBuckets').jstree({
62.       'core': {
63.         'themes': { "icons": true },
64.         'data': {
65.           "url": '/api/forge/oss/buckets',
66.           "dataType": "json",
67.           'multiple': false,
68.           "data": function (node) {
69.             return { "id": node.id };
70.           }
71.         }
72.       },
73.       'types': {
74.         'default': {
75.           'icon': 'glyphicon glyphicon-question-sign'
76.         },
77.         '#': {
78.           'icon': 'glyphicon glyphicon-cloud'
79.         },
80.         'bucket': {
81.           'icon': 'glyphicon glyphicon-folder-open'
82.         },
```

```
        'object': {
          'icon': 'glyphicon glyphicon-file'
        }
      },
      "plugins": ["types", "state", "sort", "contextmenu"],
      contextmenu: { items: autodeskCustomMenu }
    }).on('loaded.jstree', function () {
      $('#appBuckets').jstree('open_all');
    }).bind("activate_node.jstree", function (evt, data) {
      if (data != null && data.node != null && data.node.type == 'object') {
        $("#forgeViewer").empty();
        var urn = data.node.id;
        getForgeToken(function (access_token) {
          jQuery.ajax({
            url: 'https://developer.api.autodesk.com/modelderivative/v2/designdata/' + urn + '/manifest',
            headers: { 'Authorization': 'Bearer ' + access_token },
            success: function (res) {
                    if (res.status === 'success') launchViewer(urn);
                    else $("#forgeViewer").html('The translation job still running: ' + res.progress + '. Please try again in a moment.');
                  },
                  error: function (err) {
                    var msgButton = 'This file is not translated yet! ' +
                      '<button class="btn btn-xs btn-info" onclick="translateObject()"><span class="glyphicon glyphicon-eye-open"></span> ' +
                      'Start translation</button>'
                    $("#forgeViewer").html(msgButton);
                  }
                });
              })
            }
          });
        }

        function autodeskCustomMenu(autodeskNode) {
          var items;

          switch (autodeskNode.type) {
            case "bucket":
              items = {
                uploadFile: {
                  label: "Upload file",
                  action: function () {
                    uploadFile();
                  },
                  icon: 'glyphicon glyphicon-cloud-upload'
                }
              };
              break;
            case "object":
              items = {
                translateFile: {
                  label: "Translate",
                  action: function () {
```

```
                    var treeNode = $('#appBuckets').jstree(true).get_selected(true)[0]
;
                    translateObject(treeNode);
                  },
                  icon: 'glyphicon glyphicon-eye-open'
                }
              };
              break;
          }

          return items;
        }

        function uploadFile() {
          $('#hiddenUploadField').click();
        }

        function translateObject(node) {
          $("#forgeViewer").empty();
          if (node == null) node = $('#appBuckets').jstree(true).get_selected(true)[0]
;
          var bucketKey = node.parents[0];
          var objectKey = node.id;
          jQuery.post({
            url: '/api/forge/modelderivative/jobs',
            contentType: 'application/json',
            data: JSON.stringify({ 'bucketKey': bucketKey, 'objectName': objectKey }),

            success: function (res) {
              $("#forgeViewer").html('Translation started! Please try again in a momen
t.');
            },
          });
        }

```

ForgeViewer.js

This file will handle the Viewer initialization.

```
var viewer;
var addCompleted;
var addActive;
var addPlanned;
var countModifiedProgressElement;
var modelUrn;
var allElementIds;
function launchViewer(urn) {
  var options = {
    env: 'AutodeskProduction',
    getAccessToken: getForgeToken
  };
  modelUrn = urn;
  Autodesk.Viewing.Initializer(options, () => {
    viewer = new Autodesk.Viewing.GuiViewer3D(document.getElementById('forgeViewer'),

```

```
    { extensions: [ 'Autodesk.DocumentBrowser', 'MyCustomMenuExtension', 'ShowProgressExtension'] }
    );
    viewer.start();
    var documentId = 'urn:' + urn;
    Autodesk.Viewing.Document.load(documentId, onDocumentLoadSuccess, onDocumentLoadFailure);
  });
}

function onDocumentLoadSuccess(doc) {
  addCompleted = [];
  addActive = [];
  addPlanned = [];
  allElementIds = [];
  countModifiedProgressElement = 0;
  var viewables = doc.getRoot().getDefaultGeometry();
  viewer.loadDocumentNode(doc, viewables).then(i => {
    // documented loaded, any action?
  });
}

function onDocumentLoadFailure(viewerErrorCode) {
  console.error('onDocumentLoadFailure() - errorCode:' + viewerErrorCode);
}

//2
function getForgeToken(callback) {
  fetch('/api/forge/oauth/token').then(res => {
    res.json().then(data => {
      callback(data.access_token, data.expires_in);
    });
  });
}
```

Now, we have finished creating the code for displaying models in a web viewer.

3.2. Update the objects' property

This section will focus on how to add, save, and update custom properties to models' objects.

First, we need to define which properties will be added to objects

/public/js/ForgeViewer.js Line 2-4:

```
2.  var addCompleted;
3.  var addActive;
4.  var addPlanned;
```

/public/js/ForgeViewer.js Line 24-34:

```
24. function onDocumentLoadSuccess(doc) {
25.   addCompleted = [];
26.   addActive = [];
27.   addPlanned = [];
28.   allElementIds = [];
29.   countModifiedProgressElement = 0;
30.   var viewables = doc.getRoot().getDefaultGeometry();
31.   viewer.loadDocumentNode(doc, viewables).then(i => {
32.     // documented loaded, any action?
33.   });
34. }
```

We have 3 variable values: "**Planned**", "**Active**" and "**Completed**" represent the status of each object based on the schedule.

a. View Extension

Extensions provide a mechanism to write custom code that interacts with the **Viewer**. Each extension should register itself with the extension manager, providing a unique string ID which is then used to load or unload the extension during runtime.

This step of the tutorial describes the basic skeleton of an extension with a toolbar button, which triggers a code inside **Right click** function.

Let's get started, each extension should be a JavaScript file and implement, at least, the **.load** and **.unload** functions.

Open **/js/Autodesk.ADN.Viewing.Extension.ShowProgess.js**

This file adds **'Show Status'** and **'Save Status'** to the toolbar.

```
1.  class ShowProgressExtension extends Autodesk.Viewing.Extension {
```

```
2.      constructor(viewer, options) {
3.          super(viewer, options);
4.          this._group = null;
5.          this._button = null;
6.      }
7.
8.      load() {
9.          console.log('ShowProgressExtension has been loaded');
10.         return true;
11.     }
12.
13.     unload() {
14.         // Clean our UI elements if we added any
15.         if (this._group) {
16.             this._group.removeControl(this._button);
17.             if (this._group.getNumberOfControls() === 0) {
18.                 this.viewer.toolbar.removeControl(this._group);
19.             }
20.         }
21.         console.log('ShowProgressExtension has been unloaded');
22.         return true;
23.     }
24.
25.     onToolbarCreated() {
26.         // Create a new toolbar group if it doesn't exist
27.         this._group = this.viewer.toolbar.getControl('progressToolbar');
28.         if (!this._group) {
29.             this._group = new Autodesk.Viewing.UI.ControlGroup('progressToolbar');
30.             this.viewer.toolbar.addControl(this._group);
31.         }
32.
33.         // Add a new button to the toolbar group
34.         this._button = new Autodesk.Viewing.UI.Button('ShowProgressButton');
35.         this._button.onClick = (ev) => {
36.             // Execute an action here
37.             $('#dashboard').remove();
38.             loadProgress();
39.         };
40.         this._button.setToolTip('Show Status');
41.         this._button.addClass('showProgressIcon');
42.         this._group.addControl(this._button);
43.
44.
45.         // Add a new button to the toolbar group
46.         this._button = new Autodesk.Viewing.UI.Button('UpdateProgressButton');
47.         this._button.onClick = (ev) => {
48.             // Execute an action here
49.             $('#dashboard').remove();
50.             updateProgress();
51.         };
52.         this._button.setToolTip('Save Status');
53.         this._button.addClass('saveProgressIcon');
54.         this._group.addControl(this._button);
55.     }
56. }
57.
```

```
58. Autodesk.Viewing.theExtensionManager.registerExtension('ShowProgressExtension', ShowPro
    gressExtension);
```

Open **/js/Autodesk.ADN.Viewing.Extension.MyCustomMenu.js**

This file will add or assign status to elements when users right-click on selected elements and select values: “**Planned**”, “**Active**” and “**Completed**”.

```
1.  function MyCustomMenuExtension(viewer, options) {
2.      Autodesk.Viewing.Extension.call(this, viewer, options);
3.  }
4.
5.  MyCustomMenuExtension.prototype = Object.create(Autodesk.Viewing.Extension.prototype);

6.  MyCustomMenuExtension.prototype.constructor = MyCustomMenuExtension;
7.
8.  MyCustomMenuExtension.prototype.load = function () {
9.      //console.log('MyCustomMenuExtension is loaded!');
10.     this.viewer.registerContextMenuCallback('MyCustomMenu', (menu, status) => {
11.         if (status.hasSelected) {
12.             menu.push({
13.                 title: 'Add Completed',
14.                 target: () => {
15.
16.                     const selSet = this.viewer.getSelection();
17.
18.                     for (let i = 0; i < selSet.length; i++) {
19.                         this.viewer.model.getProperties(selSet[i],
20.                             function (result) {
21.                                 //console.log(selSet[i]);
22.                                 //console.log(result);
23.                                 addCompleted.push(
24.                                     {
25.                                         "UniqueId": result.externalId
26.                                     });
27.                             },
28.                             function () {
29.                                 console.log("Error");
30.                             }
31.                         );
32.                     };
33.                 }
34.             });
35.             menu.push({
36.                 title: 'Add Active',
37.                 target: () => {
38.                     const selSet = this.viewer.getSelection();
39.                     for (let i = 0; i < selSet.length; i++) {
40.                         this.viewer.model.getProperties(selSet[i],
41.                             function (result) {
42.                                 addActive.push({
43.                                     "UniqueId": result.externalId
44.                                 });
45.                             },
```

```
                                function () {
                                    console.log("Error");
                                }
                            );
                        };
                    }
                });

                menu.push({
                    title: 'Add Planned',
                    target: () => {
                        const selSet = this.viewer.getSelection();
                        for (let i = 0; i < selSet.length; i++) {
                            this.viewer.model.getProperties(selSet[i],
                                function (result) {
                                    addPlanned.push({
                                        "UniqueId": result.externalId
                                    });
                                },
                                function () {
                                    console.log("Error");
                                }
                            );
                        };
                    }
                });
            }
            else {
                menu.push({
                    title: 'Reset color',
                    target: () => {
                        this.viewer.clearThemingColors();
                    }
                });
            }
        });
        console.log('MyCustomMenuExtension is now loaded!');
        return true;
};


MyCustomMenuExtension.prototype.unload = function () {
    //alert('MyCustomMenuExtension is now unloaded!');
    this.viewer.unregisterContextMenuCallback('MyChangingColorMenuItems');
    console.log('MyCustomMenuExtension is now unloaded!');
    return true;
};

Autodesk.Viewing.theExtensionManager.registerExtension('MyCustomMenuExtension', MyCusto
mMenuExtension);
```

b. Load the extension

When the extension skeleton is ready, open the /index.html and see the added following line (which loads the file)

Open **/index.html** Line 20-21

```
20. <script src="js/Autodesk.ADN.Viewing.Extension.MyCustomMenu.js"></script>
21. <script src="/js/Autodesk.ADN.Viewing.Extension.ShowProgress.js"></script>
```

Finally we need to tell the Viewer to load the extension, in the **/public/js/ForgeViewer.js** find the following line:

```
15. viewer = new Autodesk.Viewing.GuiViewer3D(document.getElementById('forgeViewer'),
16.     { extensions: [ 'Autodesk.DocumentBrowser', 'MyCustomMenuExtension', 'ShowProgressE
    xtension'] }
17.     );
```

At this point, the extension should load and the toolbar button will show, but it doesn't execute anything until you create the behind function. This is the basic skeleton you can use to create your extensions.

c. Save added data

The important file to execute "**Save Status**" function after right-click and assign value to element is **/routes/saveFile.js**

This file will save/get data to/from server-side. In this case we are using **local file** to save added data.

You can use other methods such as MySQL database, SQL server, MongoDB or Design Automation API.

```
1.  const fs = require('fs');
2.  const express = require('express');
3.
4.  let router = express.Router();
5.
6.  // Middleware for obtaining a token for each request.
7.  router.use(async (req, res, next) => {
8.      next();
9.  });
10.
11. router.get('/getProgress', async (req, res, next) => {
12.     try {
13.         if(req.query.ModelId === null){
14.             res.status(400).end("Invalid model id!");
15.         }
16.         else{
17.             let path = 'progressCache/' + req.query.ModelId + '.json';
18.
19.             if(fs.existsSync(path)){
20.                 let rawdata = fs.readFileSync(path);
21.                 res.status(200).end(JSON.stringify(JSON.parse(rawdata)));
22.             }
23.             else{
```

```
                res.status(400).end();
            }
        }


    } catch(err) {
        next(err);
    }
});

router.post('/saveProgress', async (req, res, next) => {
    try {

        let path = 'progressCache/' + req.body.ModelId + '.json';


        if(fs.existsSync(path)){
            let rawdata = fs.readFileSync(path);
            let existingData = JSON.parse(rawdata);
            let completed = req.body.AddCompleted;
            let active = req.body.AddActive;
            let planned = req.body.AddPlanned;
            if(completed !== null && completed.length >= 0)
            {

                existingData.AddCompleted = existingData.AddCompleted.concat(completed)
;
                existingData.AddCompleted = getUnique(existingData.AddCompleted, 'Uniqu
eId');

            }
            if(active !== null && active.length >= 0)
            {
                existingData.AddActive = existingData.AddActive.concat(active);

                existingData.AddActive = getUnique(existingData.AddActive, 'UniqueId');

            }
            if(planned !== null && planned.length >= 0)
            {
                existingData.AddPlanned = existingData.AddPlanned.concat(planned);

                existingData.AddPlanned = getUnique(existingData.AddPlanned, 'UniqueId'
);
            }
            fs.writeFileSync(path, JSON.stringify(existingData));
        }
        else{
            fs.writeFileSync(path, JSON.stringify(req.body));
        }


        res.status(200).end();
    } catch(err) {
        console.log(err);
        next(err);
    }
```

```
77. });
78.
79. function getUnique(arr, comp) {
80.
81.     // store the comparison  values in array
82.     const unique =  arr.map(e => e[comp])
83.
84.   // store the indexes of the unique objects
85.     .map((e, i, final) => final.indexOf(e) === i && i)
86.
87.   // eliminate the false indexes & return unique objects
88.     .filter((e) => arr[e]).map(e => arr[e]);
89.
90. return unique;
91. }
92.
93. module.exports = router;
```

d. Getting current data

Open **/public/js/CustomJS.js**

This file to call **"Show Status"** function when a users interact with the view extension

```
1.    function updateProgress() {
2.      jQuery.post({
3.        url: '/api/forge/saveFile/saveProgress',
4.        contentType: 'application/json',
5.        data: JSON.stringify({
6.          ModelId: modelUrn,
7.          AddCompleted: addCompleted,
8.          AddActive: addActive,
9.          AddPlanned: addPlanned
10.     }),
11.       success: function (res) {
12.         alert('Successfully saved!');
13.         $('#dashboard').empty();
14.         loadProgress();
15.       },
16.       error: function (err) {
17.         console.log(err);
18.       }
19.     });
20.   }
21.
22.   function loadProgress() {
23.
24.     var data =
25.     {
26.         ModelId: modelUrn
27.     };
28.
29.     jQuery.get({
30.         url: '/api/forge/saveFile/getProgress',
31.         type: 'GET',
32.         data: data,
33.         dataType: "json",
```

```
        error: function (er) {
            console.log("Can not request");
            //console.log(er);
            alert('No progress has been saved before.')
        }
    })
    .then(res => showProgressElements(res));
}

function showProgressElements(progressData) {
    let completedIds = progressData.AddCompleted.map(a => a.UniqueId);
    let plannedIds = progressData.AddPlanned.map(a => a.UniqueId);
    let activeIds = progressData.AddActive.map(a => a.UniqueId);

    new Dashboard(NOP_VIEWER, [
        new PieChart('progress'),
        new BarChart('progress')
    ],
    progressData)

    let rtIds = getAllIds();
    setThemingColorIds(rtIds, "#bdc3c7");
    for (let i = 0; i < rtIds.length; i++) {
        viewer.model.getProperties(
            rtIds[i],
            function (result) {
                if (completedIds.includes(result.externalId)) {
                    setThemingColorId(result.dbId, "#27ae60");
                }
                else if (activeIds.includes(result.externalId)) {
                    setThemingColorId(result.dbId, "#c0392b");
                }
                else if (plannedIds.includes(result.externalId))
                {
                    setThemingColorId(result.dbId, "#2c3e50");
                }
            },
            function () {
                console.log("Error");
            }
        );
    }

    // showProgressLegend();
}

function setThemingColorIds(dbIds, colorHex) {
    let colorRGB = hexToRgb(colorHex);
    let colorVector = new THREE.Vector4(colorRGB.r / 255, colorRGB.g / 255, colorRGB.b
/ 255, 1);

    for (let i = 0; i < dbIds.length; i++) {
        viewer.setThemingColor(dbIds[i], colorVector);
    }
}

function setThemingColorId(dbId, colorHex) {
```

```
90.     let colorRGB = hexToRgb(colorHex);
91.     let colorVector = new THREE.Vector4(colorRGB.r / 255, colorRGB.g / 255, colorRGB.b
    / 255, 1);
92.
93.     viewer.setThemingColor(dbId, colorVector);
94. }
95.
96. function hexToRgb(hex) {
97.     // Expand shorthand form (e.g. "03F") to full form (e.g. "0033FF")
98.     var shorthandRegex = /^#?([a-f\d])([a-f\d])([a-f\d])$/i;
99.     hex = hex.replace(shorthandRegex, function (m, r, g, b) {
100.                return r + r + g + g + b + b;
101.            });
102.
103.            var result = /^#?([a-f\d]{2})([a-f\d]{2})([a-f\d]{2})$/i.exec(hex);
104.            return result ? {
105.                r: parseInt(result[1], 16),
106.                g: parseInt(result[2], 16),
107.                b: parseInt(result[3], 16)
108.            } : null;
109.        }
110.
111.        function getAllIds() {
112.            if (allElementIds !== null && allElementIds.length > 0) {
113.                return allElementIds;
114.            }
115.            else {
116.                allElementIds = getAllLeafIdsOfParentId(viewer.model.getData().instanceT
    ree.getRootId());
117.                return allElementIds;
118.            }
119.        }
120.
121.        function getAllLeafIdsOfParentId(id) {
122.            let allIds = [];
123.            const instanceTree = viewer.model.getData().instanceTree;
124.            if (instanceTree.getChildCount(id) > 0) {
125.                allIds.push(id);
126.                instanceTree.enumNodeChildren(id, function (child) {
127.                    allIds = allIds.concat(getAllLeafIdsOfParentId(child));
128.                }, false);
129.            }
130.            else {
131.                //console.log(id);
132.                allIds.push(id);
133.            }
134.
135.            return allIds;
136.        }
```

3.3. Dashboard

This section will guide you on how to develop a dashboard by retrieving data and creating some charts.

a. Adjust layout

This step uses the basic layout of your application, but adds an extra column for charts.

Open the **Dashboard.js** file under /public/js/dashboard/:

```
1.  // Handles the Dashboard panels
2.  class Dashboard {
3.      constructor(viewer, panels, progressData) {
4.          //var _this = this;
5.          this._viewer = viewer;
6.          this._panels = panels;
7.          this._progressData = progressData;
8.          this.adjustLayout();
9.          // this._viewer.addEventListener(Autodesk.Viewing.GEOMETRY_LOADED_EVENT, (viewe
    r) => {
10.         //     _this.loadPanels();
11.         // });
12.     }
13.
14.     adjustLayout() {
15.         // this function may vary for layout to layout...
16.         // for learn forge tutorials, let's get the ROW and adjust the size of the
17.         // columns so it can fit the new dashboard column, also we added a smooth trans
    ition css class for a better user experience
18.         var row = $(".row").children();
19.         $(row[0]).removeClass('col-sm-4').addClass('col-sm-2 transition-width');
20.         $(row[1]).removeClass('col-sm-8').addClass('col-sm-7 transition-
    width').after('<div class="col-sm-3 transition-width" id="dashboard"></div>');
21.         //console.log("adjustLayout");
22.         this.loadPanels();
23.         this._viewer.resize();
24.     }
25.
26.     loadPanels () {
27.         var _this = this;
28.         var data = new ModelData(this._viewer, this._progressData);
29.         data.init(function () {
30.             $('#dashboard').empty();
31.             _this._panels.forEach(function (panel) {
32.                 // let's create a DIV with the Panel Function name and load it
33.                 panel.load('dashboard', viewer, data);
34.             });
35.         });
36.     }
37. }
```

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

Let's also add a couple extra CSS classes to help on the layout.
Open **/css/main.css** file Line 25-38:

```
25. #dashboard{
26.   overflow: auto;
27.   height: calc(100vh - 100px);
28. }
29.
30. .transition-width {
31.   transition: width 1s ease-in-out;
32. }
33.
34. .dashboardPanel {
35.   width: 100%;
36.   padding: 3%;
37.   display: block;
38. }
```

b. Retrieving the data

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

Open **/public/js/dashboard/DashboardPanel.js**. This file prepares data for creating charts

```
1.  // Dashboard panel base
2.  class DashboardPanel {
3.      load(parentDivId, divId, viewer) {
4.          this.divId = divId;
5.          this.viewer = viewer;
6.          $('#' + parentDivId).append('<div id="' + divId + '" class="dashboardPanel"></d
    iv>');
7.      }
8.  }
9.
10. // Dashboard panels for charts
11. class DashboardPanelChart extends DashboardPanel {
12.     load(parentDivId, divId, viewer, modelData) {
13.         if (this.propertyToUse !== 'progress' && !modelData.hasProperty(this.propertyTo
    Use)){
14.             alert('This model does not contain a ' + this.propertyToUse +' property for
     the ' + this.constructor.name);
15.             console.log('These are the properties available on this model: ');
16.             console.log(Object.keys(modelData._modelData));
17.             return false;
18.         }
19.         divId = this.propertyToUse.replace(/[^A-Za-z0-
    9]/gi, '') + divId; // div name = property + chart type
20.         super.load(parentDivId, divId, viewer);
21.         this.canvasId = divId + 'Canvas';
```

```
22.         $('#' + divId).append('<canvas id="' + this.canvasId + '" width="400" height="4
    00"></canvas>');
23.         this.modelData = modelData;
24.         return true;
25.     }
26.
27.     generateColors(count) {
28.         var background = []; var borders = [];
29.         for (var i = 0; i < count; i++) {
30.             var r = Math.round(Math.random() * 255); var g = Math.round(Math.random() *
     255); var b = Math.round(Math.random() * 255);
31.             background.push('rgba(' + r + ', ' + g + ', ' + b + ', 0.2)');
32.             borders.push('rgba(' + r + ', ' + g + ', ' + b + ', 0.2)');
33.         }
34.         return { background: background, borders: borders };
35.     }
36. }
37.
38. // Model data in format for charts
39. class ModelData {
40.     constructor(viewer, progressData) {
41.         this._modelData = {};
42.         this._viewer = viewer;
43.         this._progressData = progressData;
44.     }
45.
46.     init(callback) {
47.         var _this = this;
48.         let completedIds = _this._progressData.AddCompleted.map(a => a.UniqueId);
49.         let activeIds = _this._progressData.AddActive.map(a => a.UniqueId);
50.         let plannedIds = _this._progressData.AddPlanned.map(a => a.UniqueId);
51.
52.         _this._modelData['progress'] = [];
53.         _this._modelData['progress']['completed']= [];
54.         _this._modelData['progress']['planned']= [];
55.         _this._modelData['progress']['active']= [];
56.
57.         _this.getAllLeafComponents(function (dbIds) {
58.             var count = dbIds.length;
59.             dbIds.forEach(function (dbId) {
60.                 viewer.getProperties(dbId, function (props) {
61.
62.                     if (completedIds.includes(props.externalId)) {
63.                         _this._modelData['progress']['completed'].push(dbId);
64.                     }
65.                     else if (activeIds.includes(props.externalId)) {
66.                         _this._modelData['progress']['active'].push(dbId);
67.                     }
68.                     else if (plannedIds.includes(props.externalId))
69.                     {
70.                         _this._modelData['progress']['planned'].push(dbId);
71.                     }
72.
73.                     props.properties.forEach(function (prop) {
74.                         if (!isNaN(prop.displayValue)) return; // let's not categorize
    properties that store numbers
75.
```

```
                        // some adjustments for revit:
                        prop.displayValue = prop.displayValue.replace('Revit ', ''); // remove this Revit prefix
                        if (prop.displayValue.indexOf('<') == 0) return; // skip categories that start with <

                        // ok, now let's organize the data into this hash table
                        if (_this._modelData[prop.displayName] == null) _this._modelData[prop.displayName] = {};
                        if (_this._modelData[prop.displayName][prop.displayValue] == null) _this._modelData[prop.displayName][prop.displayValue] = [];
                        _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-nodes
        viewer.getObjectTree(function (tree) {
            var leaves = [];
            tree.enumNodeChildren(tree.getRootId(), function (dbId) {
                leaves.push(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[propertyName][key].length);
        }

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

c. Add charts

There are many libraries to create charts, in this sample we use **Chart.js**, a very simple yet nice library to use 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>*

Open **/public/index.html** Line 25-27:

```
25. <!--Chart JS  packages-->
26. <script src="//cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.3/Chart.min.js" ></script>
27. <link rel="stylesheet" href="//cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.3/Chart.min.
    css" />
```

Then, create a Bart chart with the code in **/public/js/dashboard/PanelBarCharts.js**

```
1.  class BarChart extends DashboardPanelChart {
2.      constructor(property) {
3.          super();
4.          this.propertyToUse = property;
5.      }
6.
7.      load(parentDivId, viewer, modelData) {
8.          if (!super.load(parentDivId, this.constructor.name, viewer, modelData)) return;

9.          this.drawChart();
10.     }
11.
12.     drawChart() {
13.         var _this = this; // need this for the onClick event
14.
15.         var ctx = document.getElementById(this.canvasId).getContext('2d');
16.
17.         if(this.propertyToUse === 'progress'){
18.             var colors = {
19.                 background:['rgba(39,174,96,1)','rgba(44,62,80,1)','rgba(192,57,43,1)']
    ,
20.                 borders:['rgba(39,174,96,1)','rgba(44,62,80,1)','rgba(192,57,43,1)']
21.             };
22.             new Chart(ctx, {
23.                 type: 'bar',
24.                 data: {
25.                     labels: this.modelData.getLabels(this.propertyToUse),
26.                     datasets: [{
27.                         data: this.modelData.getCountInstances(this.propertyToUse),
28.                         backgroundColor: colors.background,
29.                         borderColor: colors.borders,
30.                         borderWidth: 1
31.                     }]
32.                 },
33.                 options: {
34.                     scales: {
35.                         yAxes: [{
36.                             ticks: {
```

```
                                beginAtZero: true
                            }
                        }]
                    },
                    legend: {
                        display: false
                    },
                    'onClick': function (evt, item) {
                        _this.viewer.isolate(_this.modelData.getIds(_this.propertyToUse
, item[0]._model.label));
                    }
                }
            });
        }
        else{
            var colors = this.generateColors(this.modelData.getLabels(this.propertyToUs
e).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]._model.label));
                    }
                }
            });
        }

    }
}
```

The procedure to create the Pie chart is the following code in **/public/js/dashboard/PanelPieCharts.js:**

```
class PieChart extends DashboardPanelChart {
    constructor(property) {
        super();
```

```
4.          this.propertyToUse = property;
5.      }
6.
7.      load(parentDivId, viewer, modelData) {
8.          if (!super.load(parentDivId, this.constructor.name, viewer, modelData)) return;

9.          this.drawChart();
10.     }
11.
12.     drawChart() {
13.         var _this = this; // need this for the onClick event
14.
15.         var ctx = document.getElementById(this.canvasId);
16.
17.         if(this.propertyToUse === 'progress'){
18.             var colors = {
19.                 background:['rgba(39,174,96,1)','rgba(44,62,80,1)','rgba(192,57,43,1)']
    ,
20.                 borders:['rgba(39,174,96,1)','rgba(44,62,80,1)','rgba(192,57,43,1)']
21.             };
22.
23.             new Chart(ctx, {
24.                 type: 'doughnut',
25.                 data: {
26.                     labels: this.modelData.getLabels(this.propertyToUse),
27.                     datasets: [{
28.                         data: this.modelData.getCountInstances(this.propertyToUse),
29.                         backgroundColor: colors.background,
30.                         borderColor: colors.borders,
31.                         borderWidth: 1
32.                     }]
33.                 },
34.                 options: {
35.                     legend: {
36.                         display: true
37.                     },
38.                     'onClick': function (evt, item) {
39.                         _this.viewer.isolate(_this.modelData.getIds(_this.propertyToUse
    , item[0]._model.label));
40.                     }
41.                 }
42.             });
43.         }
44.         else
45.         {
46.             var colors = this.generateColors(this.modelData.getLabels(this.propertyToUs
    e).length);
47.             new Chart(ctx, {
48.                 type: 'doughnut',
49.                 data: {
50.                     labels: this.modelData.getLabels(this.propertyToUse),
51.                     datasets: [{
52.                         data: this.modelData.getCountInstances(this.propertyToUse),
53.                         backgroundColor: colors.background,
54.                         borderColor: colors.borders,
55.                         borderWidth: 1
56.                     }]
```

```
57.                 },
58.                 options: {
59.                     tooltips: {
60.                         enabled: false
61.                     },
62.                     plugins: {
63.                         datalabels: {
64.                             formatter: (value, ctx) => {
65.                                 let sum = 0;
66.                                 let dataArr = ctx.chart.data.datasets[0].data;
67.                                 dataArr.map(data => {
68.                                     sum += data;
69.                                 });
70.                                 let percentage = (value*100 / sum).toFixed(2)+"%";
71.                                 return percentage;
72.                             },
73.                             color: '#fff',
74.                         }
75.                     },
76.                     legend: {
77.                         display: true
78.                     },
79.                     'onClick': function (evt, item) {
80.                         _this.viewer.isolate(_this.modelData.getIds(_this.propertyToUse
    , item[0]._model.label));
81.                     }
82.
83.                 }
84.             });
85.         }
86.     }
87. }
```

4. Run the code

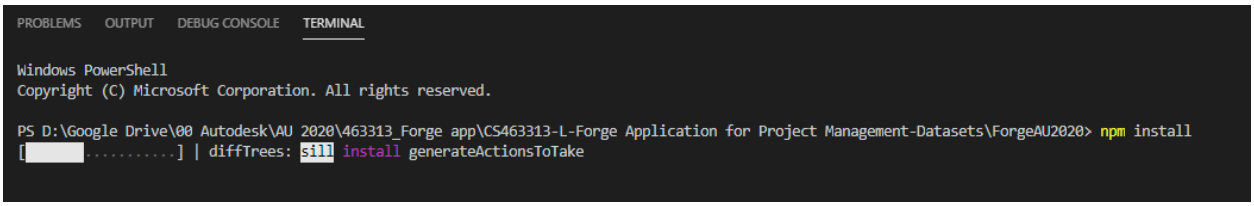
Now, we are ready to run the code sample!

Click the link below to watch the video tutorial:

https://youtu.be/yWoOKQy4MHw

Open the Terminal in Visual Code Studio, Let's install all the packages with this command:

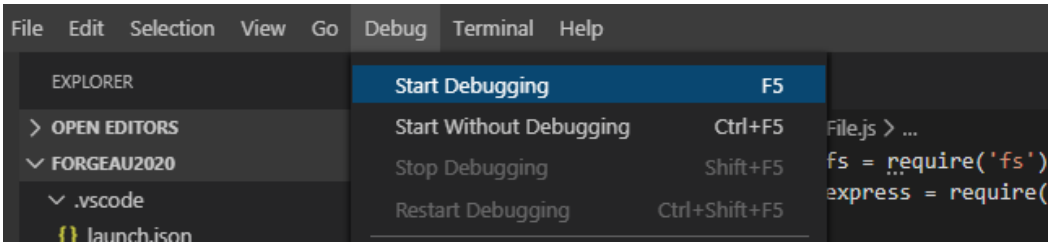
```
npm install
```



It will take a while for the packages to download and install. If the system informs you that npm is not installed, please revisit the section 2.Setup and repeat the steps.

Once the installation is done, we can start the web server.

Click on **Debug/Start Debugging** or simply **press F5**



Then, open a browser and load the URL http://localhost:3000.

Now, you can try to test your Forge application for your project management.

Conclusion

1. Summary

In today's session, we learned how to

- Set up a simple Express web server
- Two-Legged Authentication using the Forge Authentication APIs
- Create a bucket (Data Management API)
- Upload a file and save it as an object in a bucket (Data Management API)
- Read objects from a bucket (Data Management API)
- Translate the object into the SVF format (Model Derivative API)
- Display the SVF data in a web browser (Viewer API)
- Add custom properties to models' objects. (Viewer Extension)
- Save added data.
- Getting current data of models.
- Retrieve added custom data for displaying on dashboard.
- Add dashboard: bar chart and pie chart (Viewer API)

2. Next steps

2.1. Deployment

This is when your app goes live! There are several options out there, let's focus on a few.

Amazon Web Services

Microsoft Azure

Heroku

Learn more at https://learnforge.autodesk.io/#/deployment/

2.2. Support & Online resources

Online resources:

Documentation: Developer Portal is the "source of truth" on Forge!

Samples: Autodesk Forge on Github

Blog: Forge Blog

Getting help:

https://forge.autodesk.com/en/support/get-help

https://forge.autodesk.com/FAQ

Questions/concerns about this class?

Please post your comments on the class page or **email us:**

phuc.le@autodesk.com

lehieuhongphuc@gmail.com

leuyvo@gmail.com

Thank you!