AUTODESK. UNIVERSITY

CS322871

Integrating Power BI and BIM 360® Through Forge for Dynamic Construction Analytics

Marcelo Nonato Santos
Camargo Corrêa Infra

Diego Faria
Frazillio & Ferroni

Learning Objectives

- Discover the typical workflows of construction site analytics reporting and their challenges;
- Learn about the system architecture required to integrate BIM 360® and *Power BI*;
- Discover the tools and workflows used for the integration process;
- See some examples of *Power BI* analysis from BIM 360® data.

Description

This class describes how Camargo Corrêa Infra, one of the largest Brazilian construction companies, integrated its BIM 360® hub to Microsoft *Power BI*, through Forge APIs, to create dynamic graphs to monitor field information (issues and checklists).

The traditional process uses paper-based forms to register the checklists and issues, which are later digitized into spreadsheets so the necessary analysis can run. With the use of BIM 360® software, despite the digitalization happening on the source input, the data visualization does not meet the requirements of all the stakeholders.

On this scenario, we have developed an integration of BIM 360® with *Power BI* for the creation of dynamic customizable dashboards, getting real-time BIM 360® data from the field to the office. The class will approach the system architecture necessary for this integration, and useful

tips and tools to better understand and visualize data from BIM 360®, even for people who are not programmers.

Speakers

Marcelo Nonato Santos

Civil Engineer graduated from the Polytechnic School of USP, with a postgraduate degree in Project Management and in Higher Education Teaching. He has been working at Camargo Corrêa for 26 years, with experience in construction sites and in the corporate area, working in the sectors as Production, Planning, Productivity, Engineering, Project Control, Cost Competitiveness and Knowledge Management. He also has extensive experience as an internal consultant for the SAP System, for the "Project System" module. In 2014, he was invited to lead the Corporate Management of Virtual Design Construction, with the mission of consolidating the use of BIM in the company, increasing the maturity level of Camargo Corrêa BIM processes.

Diego Faria

Diego has been a BIM Consultant since 2011, located in São Paulo, Brazil. He holds a Bachelor of Engineering (Chemical Engineering) and a Master on Civil Construction, focused on procurement for BIM projects, from Poli-USP. He specializes in Revit, AutoCAD Plant 3D/P&ID, BIM 360®, BIM management and BIM procurement. He provides support for BIM implementation and training for engineering companies. Having participated in several multidisciplinary projects, Diego has faced many challenges requiring him to create tailored workflows for each companies' needs, and he is always interested in enhancing workflows with new product features. Currently, he manages the technical team at Frazillio & Ferroni for all of their project deliveries.

Definitions

REST

Representational State Transfer is a software architectural style that allows the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations.

The architecture of the channel of communication between the web application and other systems.

API

Application Programming Interface is an interface or communication protocol between a client and a server intended to simplify the building of client-side software.

A way to read information from a software without having to know how the software was built (coded). Therefore, the term REST API refers to the routines that allow information to be extracted from the web application (BIM 360®).

JSON

Javascript Object Notation is a text based lightweight data-interchange format that is easy for humans to understand and for machines to parse and generate.

The format of the information that has made available by using the REST API to extract data.

ETL

Extract, Transform, Load is a general procedure of copying data from one or more sources into a destination system, which represents the data differently from the source.

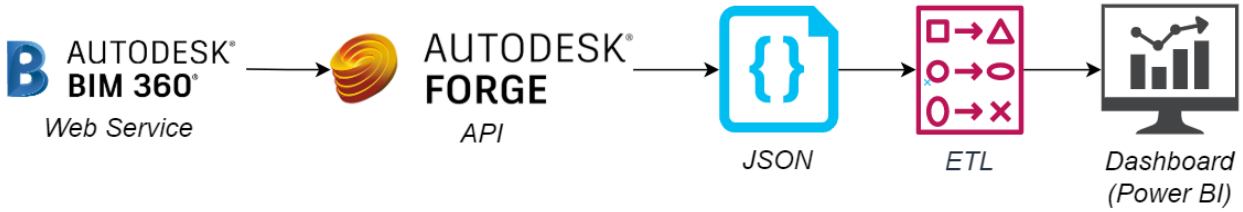
The procedures utilized to manage the data extracted from the REST APIs and use it to build a product (in our case, the dashboard).

R-code

R is a language and environment for statistical computing and graphics, with accompanying free development software. One of R's great strengths is its ease of use.

Introduction

This class aims to describe the steps towards understanding BIM 360® API and building dashboards from the data extracted from it. As the authors are not developers, we tried to explain how we matured our knowledge and the tools used throughout the process of building this solution. The process used to achieve the final product can be described by the following workflow:

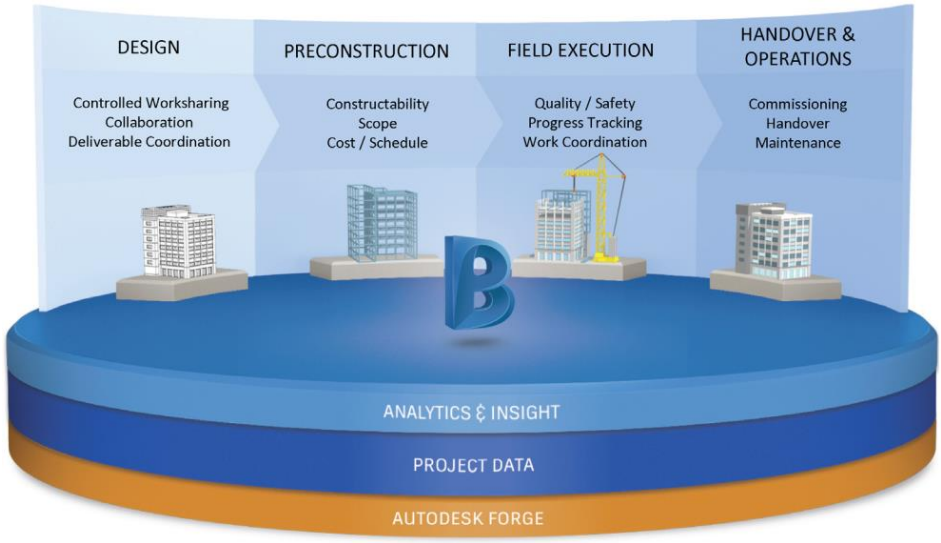


Each step of this workflow is described in this document in a simplified manner, aiming to provide a guide for professionals that want to start building their own solution.

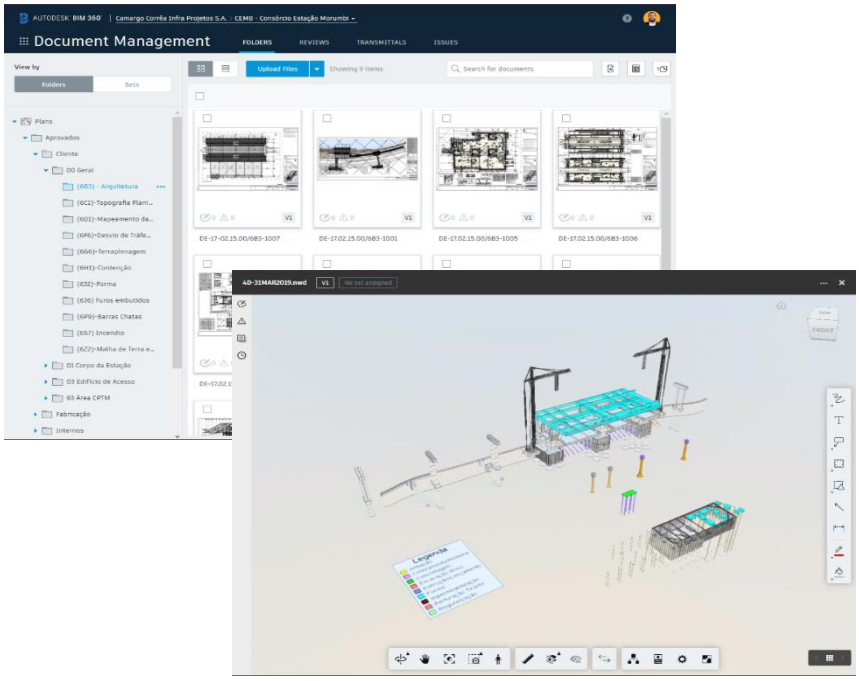
It is also important to note that the solution is a data extractor to enable integration with *Power BI*, rather than an entire application built to connect to BIM 360®.

BIM 360® Usage

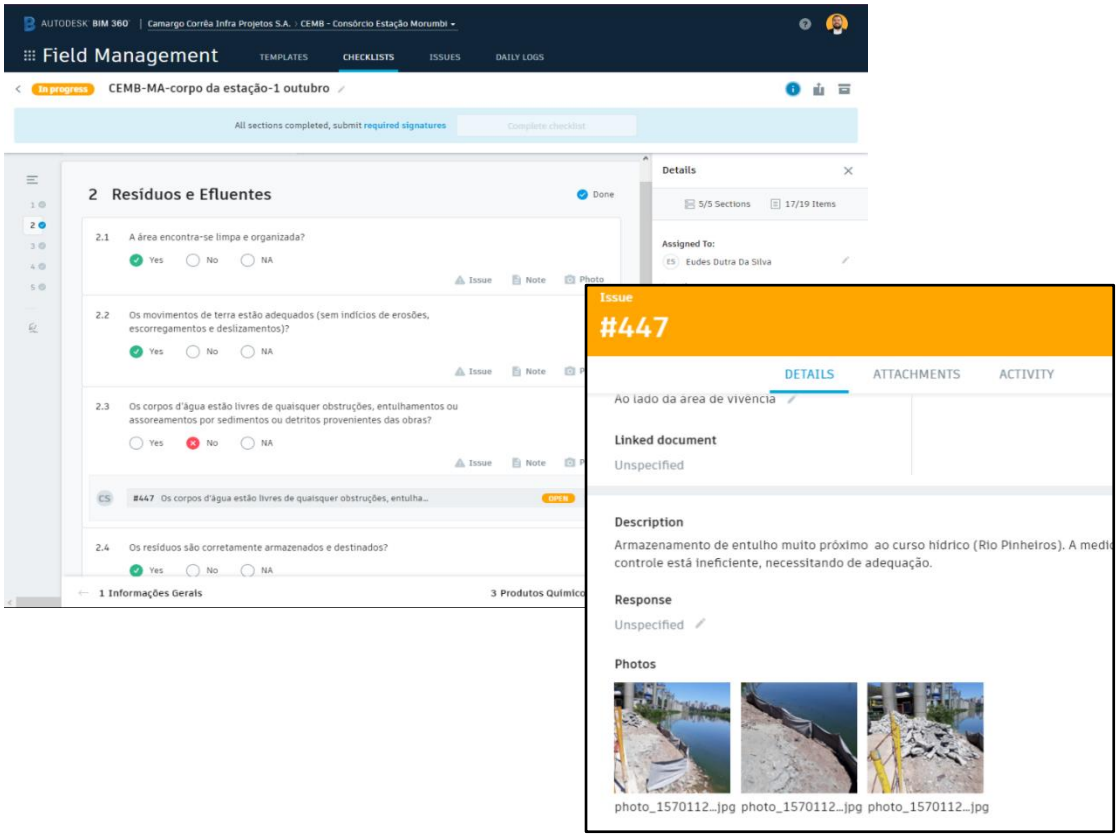
BIM 360® product ecosystem is comprised of different web applications, focused on specific project stages, available as separate modules:



The applications for the workflow described on this class are the BIM 360® Docs and BIM 360® Build. The first is the backbone of the entire BIM 360® ecosystem, enabling project data control, review and collaboration between the stakeholders.



The second is a module dedicated to field information management, such as quality checklists and field issues.

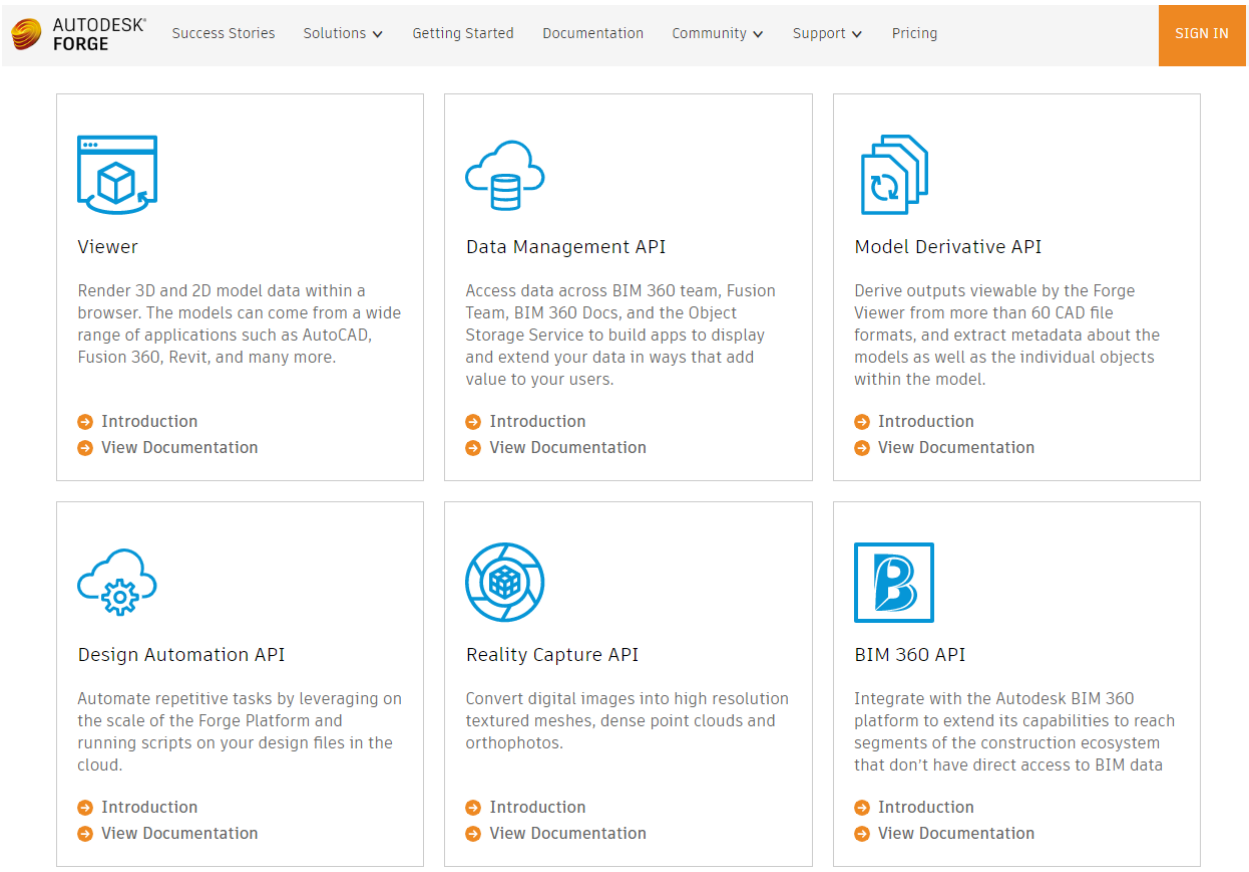


Both products are very intuitive and offer mobile access as well, which is extremely relevant in order to maintain information flowing through all of the stakeholders.

By using these two applications, the company has been able to digitize their construction sites, guaranteeing access to the latest and most accurate construction documents, eliminating paper-based exchanges. Not only that but, the indexed data feeds business intelligence dashboards, provides decision makers with real-time accurate information from the field.

Information Exchange with Autodesk® Forge

In order to get the information in and out of BIM 360®, an API is necessary. Autodesk® Forge is comprised of multiple API's, each one dedicated to a specific group of tasks related to Autodesk® cloud ecosystem. All of the API's available can be explored on forge.autodesk.com:



For the tasks described on this paper, only the BIM 360® API was used and its documentation is available on Forge website:

https://forge.autodesk.com/en/docs/bim360/v1/overview/introduction/

From the documentation portal, it is possible to explore all of endpoints available for the tasks needed for the integration process and there are very detailed tutorials for beginners. This information was invaluable throughout the entire integration process.

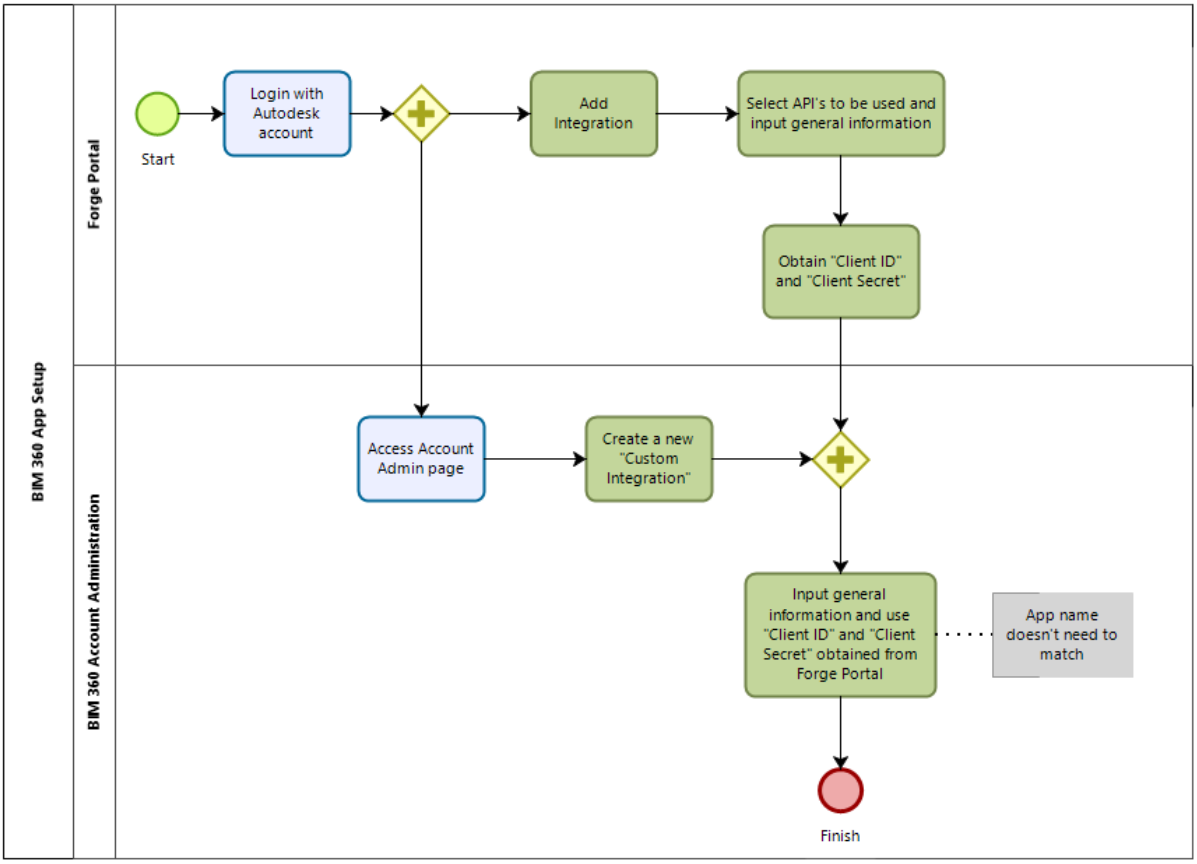
Also on the Forge webpage, it is possible to create the Forge App, which is the first step towards building an integration. This process is described on the following link:

https://forge.autodesk.com/en/docs/oauth/v2/tutorials/create-app/

After creating the app, it is also necessary to grant access for it through BIM 360® Account Admin portal, as described on the following link:

https://forge.autodesk.com/en/docs/bim360/v1/tutorials/getting-started/manage-access-to-docs/

Once these two initial settings are set up, it is possible to start getting information from and into your BIM 360® account and projects. The following diagram depicts the main steps of the workflow described on the links:



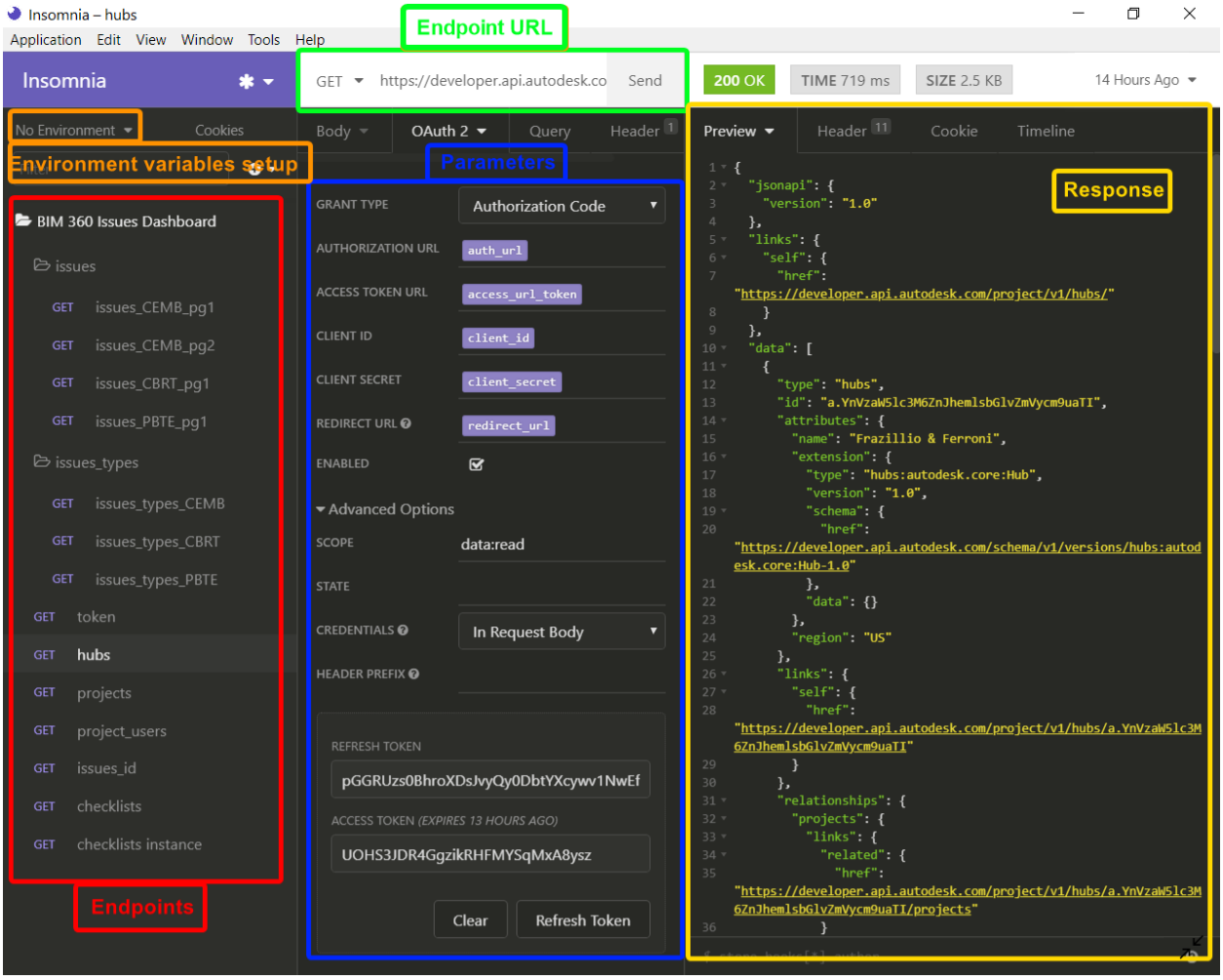
Data Acquisition from BIM 360®

REST Client (No Coding Required)

The first step towards building the integration was to understand how the information returned from BIM 360® through the Forge endpoints was structured. This way, it is possible to validate what information was present in the response JSON and adequately structure the ETL on the following steps. For this task, we chose a REST client *Insomnia* (https://insomnia.rest), so that we could easily setup endpoints and understand how to put all of the information together. This proved to be extremely useful as we were not used to JSON file structure in the first place, but

also because the modules of BIM 360® (Docs and Build) arrange the information returned differently.

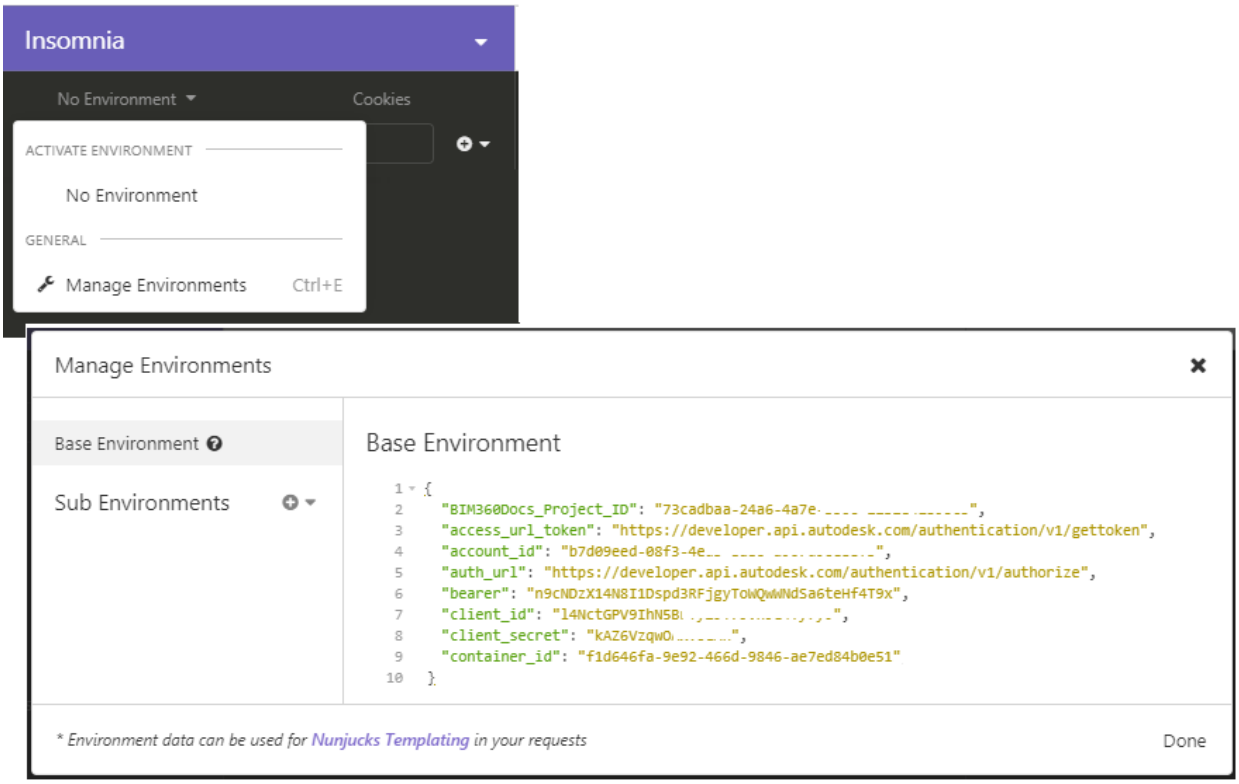
The following image shows the main sections of *Insomnia's* interface:



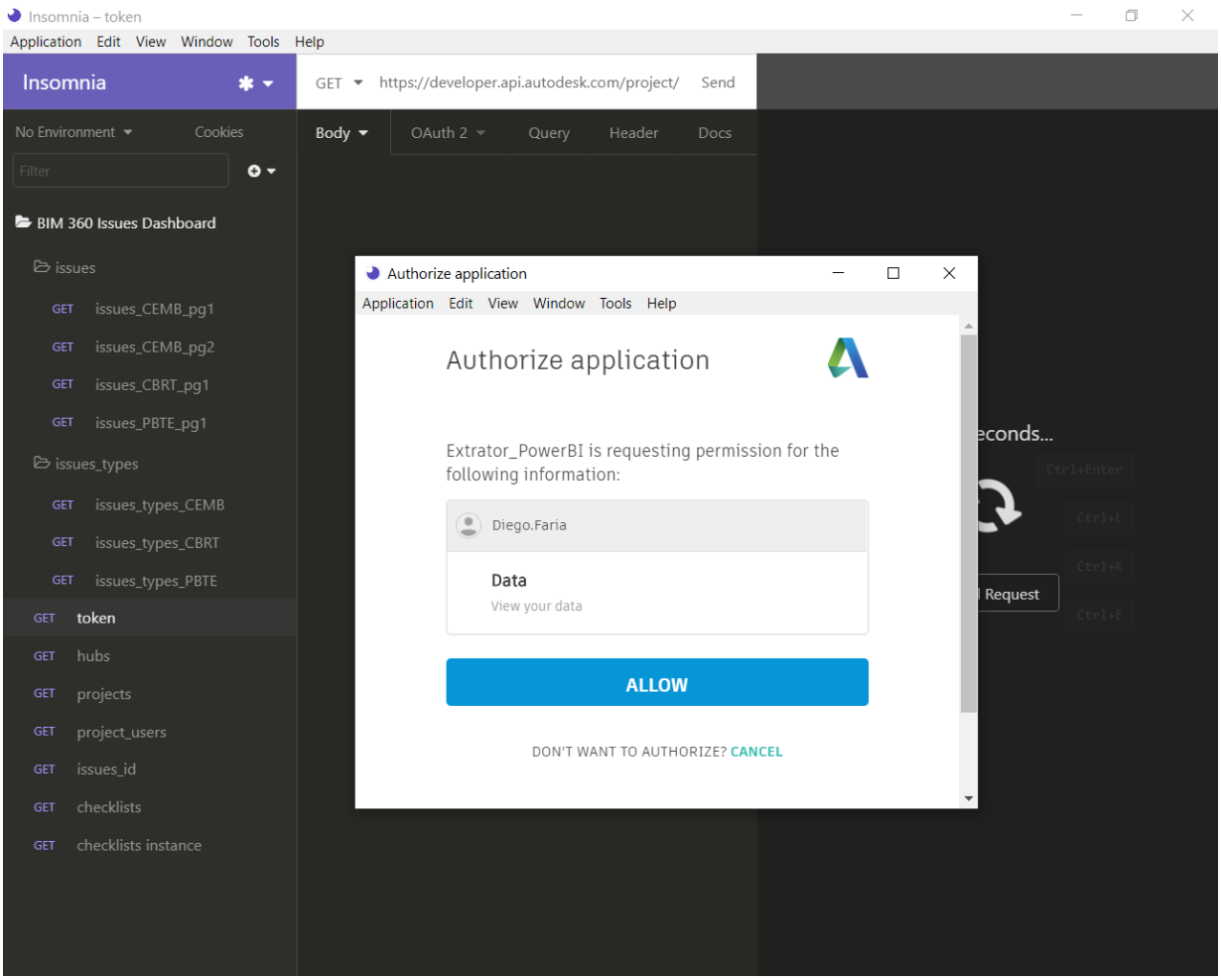
Endpoints can be stored in the *Insomnia* client (red section), each of them using a specific request URL (green section), with different parameters (blue section). The request returns a JSON response (yellow section) that can be previewed directly in the client or saved to a text file.

The environment variables setup is extremely useful in order to deal with the large alphanumeric codes used for the endpoints, especially for the OAuth parameters (the environment variables

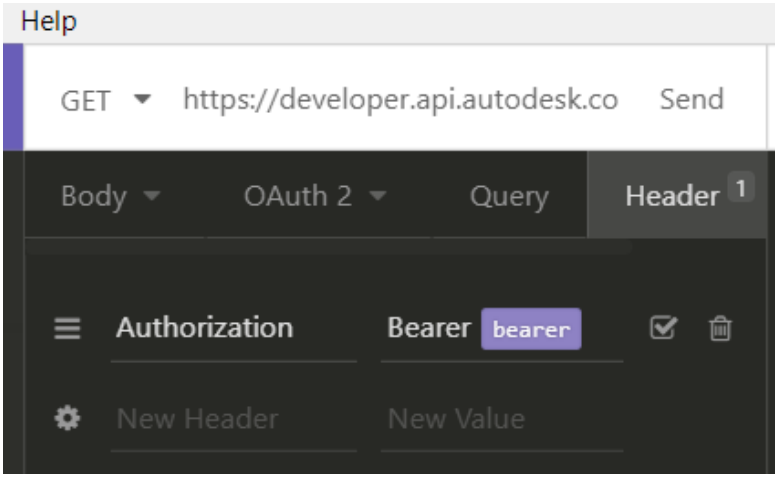
are the ones in orange in the next image). They can be easily setup (button marked in orange in the previous image), as shown in the image as follows:



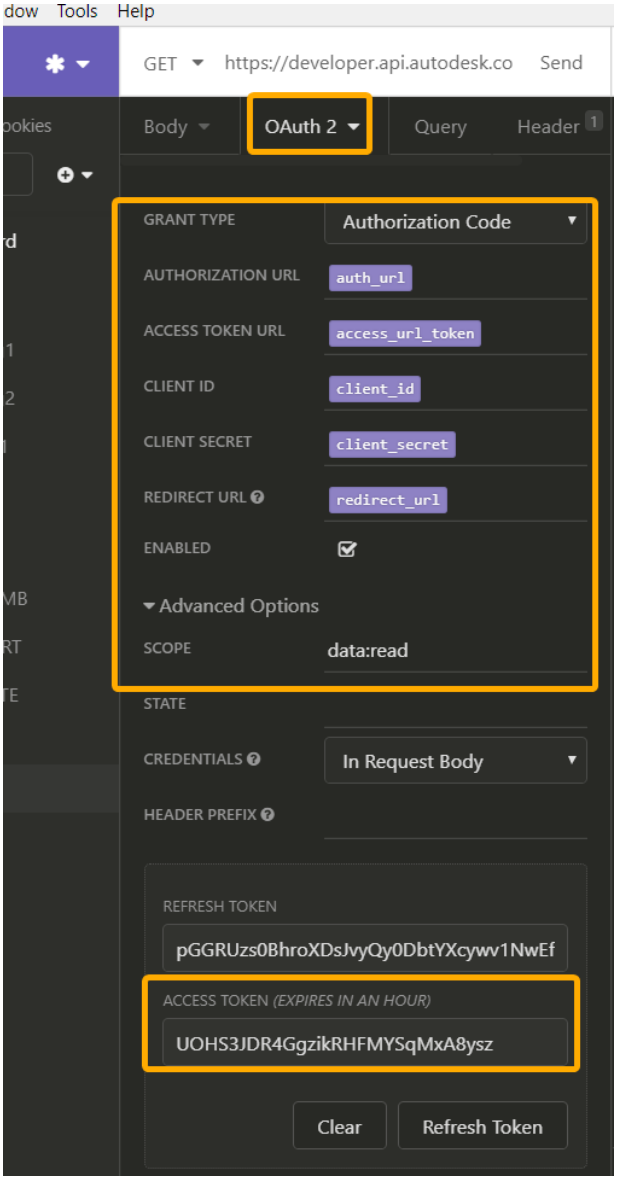
The information needed for the endpoints follows the same logic indicated on the “R-code Powering Data ETL” workflow in the next topic. The only step that remained necessary after the code implemented in R was the generation of the 3-legged token, because of the nature of its authentication. After the request URL is used, you have to manually login in order to provide access for the API to return the token, as shown in the following image:



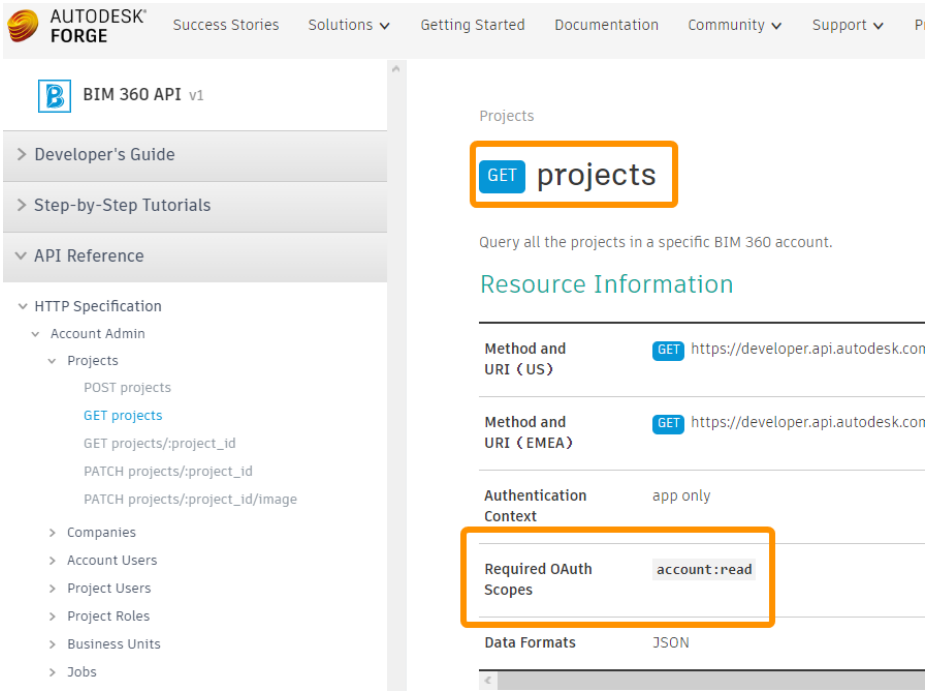
This token is used under the “Header” of every request for the “Bearer” value:



It is also important to mention that the OAuth parameters, which we strongly encourage you to create environment variables for, may need different “Scopes”.



Each endpoint has a specific scope they must match in order to work, which can be verified in the Forge documentation as shown in the image bellow. They are usually either “data:read” or “account:read” for most of the “GET” endpoints.



As mentioned before, the REST client is a very good way to start to understand how data is structured in the JSON responses and helped us to move forward to more elaborate architectures.

In the *Power BI* application, you can import the JSON file generated in *Insomnia* and make the ETL for Dashboards generation. The structure of JSON is explained in detail on the Forge webpage.

R-code Powering Data ETL

In the definition from Wikipedia, R is a programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis. Polls, data mining surveys, and studies of scholarly literature databases show substantial increases in popularity as of October 2019, R ranks 15th in the TIOBE index, a measure of popularity of programming languages.

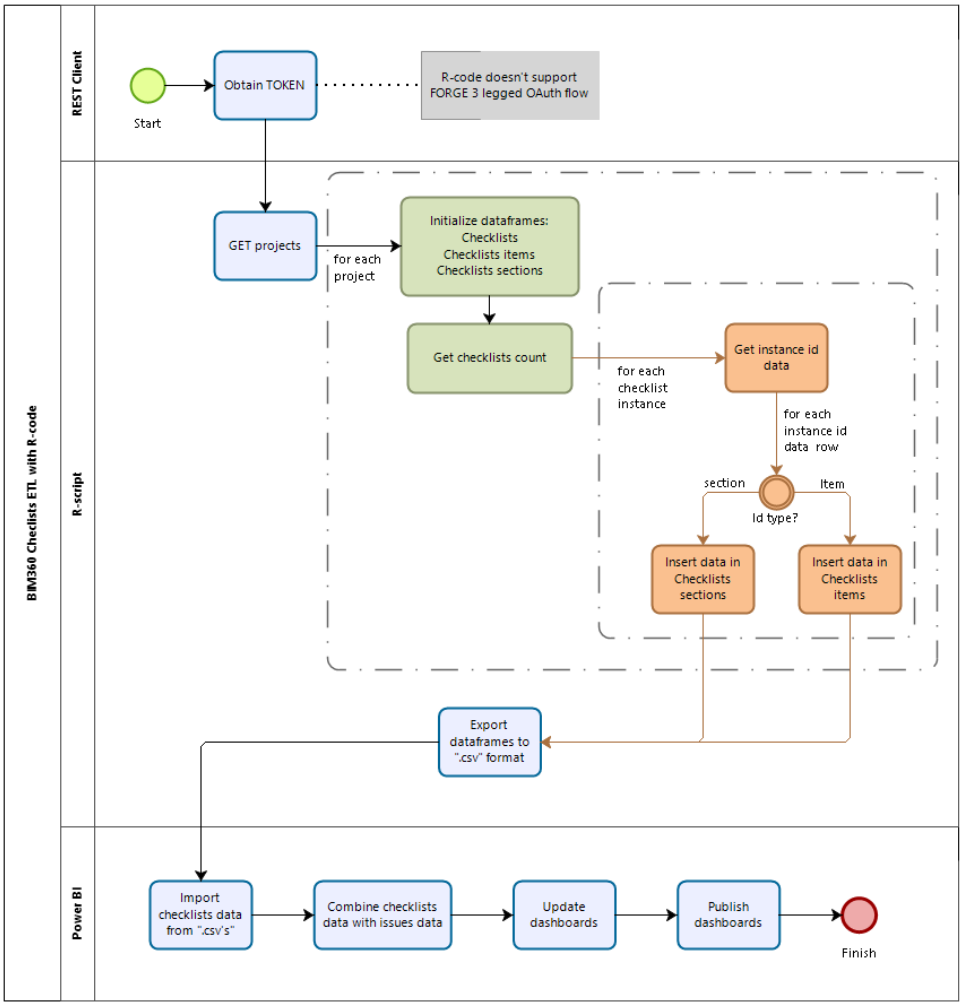
A GNU package, source code for the R software environment is written primarily in C, Fortran, and R itself and is freely available under the GNU General Public License. Pre-compiled binary versions are provided for various operating systems. Although R has a command line interface,

there are several graphical user interfaces, such as *RStudio* (https://rstudio.com/products/rstudio/download/), an integrated development environment.

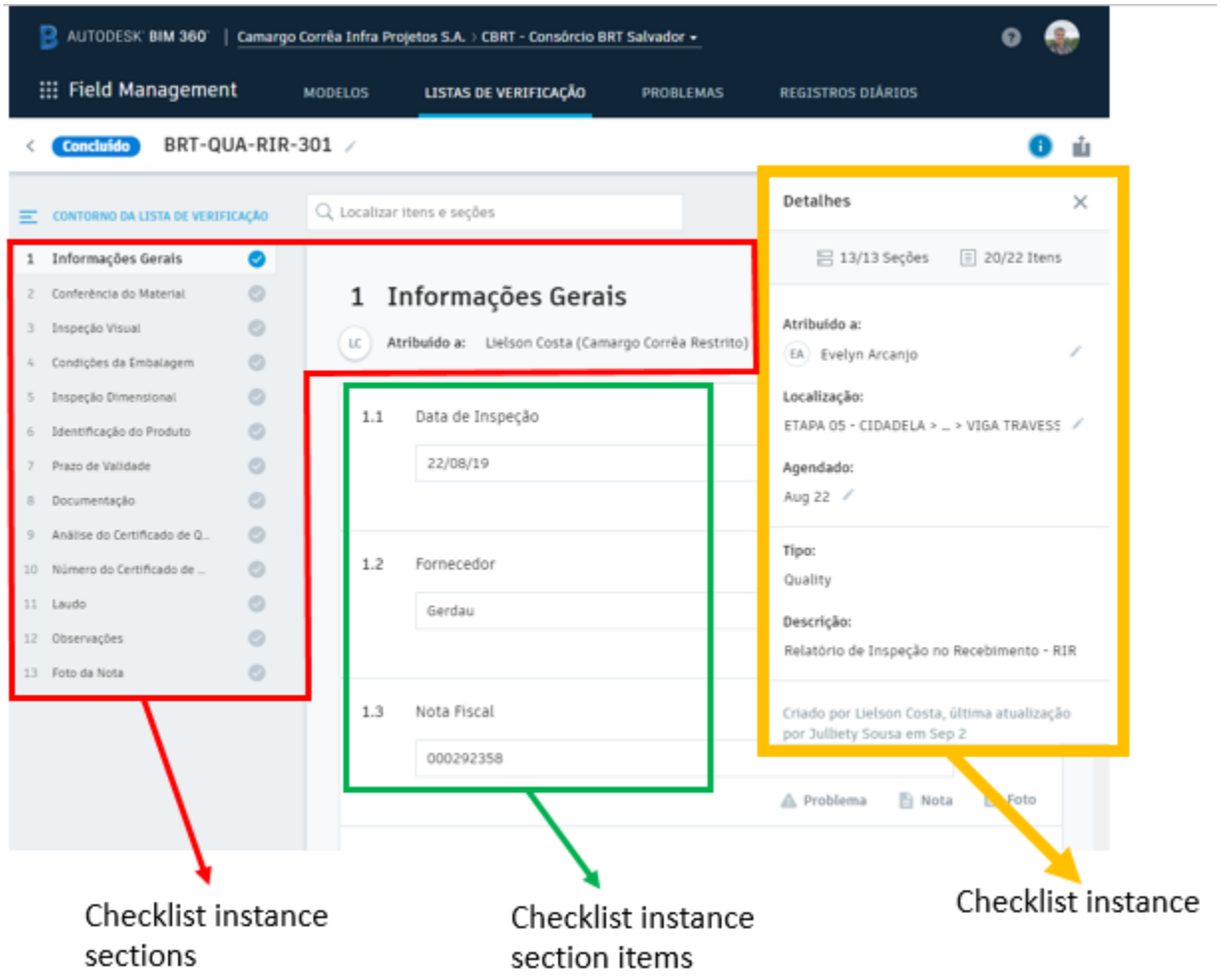
From the article published by Sam Nseir (November 2017), *Integrate BIM 360® data with Power BI using R*, we studied the use of the R language for developing information extraction scripts, where extraction would not be feasible using only REST. As an example, obtaining the details of the LVs requires the selection of data via GET command for each document, being necessary to execute the GET thousands of times, which is impracticable in a manual data extraction.

The R language is easily learned by professionals with basic programming notions, due to its simple syntax, avoiding the development of complex codes in its use.

The following diagram presents the ETL process for obtaining the checklists details aided by R language:



The image below illustrates what information in BIM 360® Build is obtained via R script:



R-code explanation

Below we present some excerpts of the script in R that operationalizes the presented flow. Some code snippets have been omitted to make ETL steps easier to understand. The full script is available at:

https://1drv.ms/u/s!AtT_VHzhvN-6iYw2W2MOT3uviOoijQ?e=X3zu9Q

GET Projects:

As the R-code use in our environment is temporary, we did not create a data selection with the GET Projects command. In this case, we simply create a vector variable with the Container_Id's from BIM 360® projects:

```
BIM360Container_Id <- c(
  #1 CBRT
  "60221808-c0c5-45d6-a829-928e036b5af1",
  #2 CEMB
  "c622751f-2c3f-4f51-bbed-fd2081d803c4",
  #3 PBTE
  "f1d646fa-9e92-466d-9846-ae7ed84b0e51",
  #4 CSBI
  "42b64763-5df1-434a-aabb-02d2aaed0981",
  #5 LT13
  "f00f3b60-447c-4f0e-82b9-fa53ca10ec92",
  #6 CFLN
  "21c260af-0cd7-4626-95cc-dd45d29d235d"
)
```

Initialize Data frames:

Each data frame is created with the fields required for analysis in *Power BI*.

```
BIM360Checklists <-
  data.frame(
    "data.relationships.container.data.id" = character(),
    "data.id" = character(),
    "data.attributes.scheduledDate" = character(),
    "data.attributes.location" = character(),
    "data.attributes.title" = character(),
    "data.attributes.createdAt" = character(),
    "data.attributes.createdBy" = character(),
    "data.attributes.completedItemsCount" = character(),
    "data.attributes.progress" = character(),
    "data.attributes.status.name" = character(),
    "data.relationships.createdFrom.data.id" = character()
  )

BIM360Checklists_included_items <-
  data.frame(
    "instance_id" = character(),
    "included.id" = character(),
    "included.attributes.modifiedBy" = character(),
    "included.attributes.updatedAt" = character(),
    "included.attributes.status" = character(),
    "included.attributes.title" = character(),
```

```
    "included.attributes.number" = character(),
    "included.attributes.answer" = character(),
    "included.attributes.isRequired" = character(),
    "included.attributes.responseType.name" = character()
  )

BIM360Checklists_included_sections <-
  data.frame(
    "id" = character(),
    "instance_id" = character(),
    "included.attributes.status" = character(),
    "included.attributes.title" = character(),
    "included.attributes.number" = character(),
    "data.attributes.instructions" = character()
   )
```

Get checklists instances count:

At this point we get the checklist count for use in the data extraction loop.

```
#Load libraries required for the R script
library("httr")
library("jsonlite")
library("httpuv")
library("xml2")

#Init connection variables
App_Client_ID <- "l4NctGPV9IhN5BrYyZ9T79V....."
App_Client_Secret <- "kAZ6VzqwOA....."
BIM360_Account_ID <- "b7d09eed-08f3-4ed3-8a86-a90....."
Redirect_Uri <- https://oauth.powerbi.com/views/oauthredirect.html

Get_Checklists_URL <-
  Paste("https://developer.api.autodesk.com/bim360/checklists/v1/containers/",
    BIM360Container_Id[cont],
    "/instances?page[offset]=",
    pageoffset,
    sep = ""
  )

#Command GET checklists instances
Get_Checklists_Request <-
  GET(Get_Checklists_URL,
     add_headers("Authorization" = Access_Token))

#Set JSON object to allow data extraction
Get_Checklists_Data <-
  jsonlite::fromJSON(content(Get_Checklists_Request,
```

```
                    "text",
                    "application/json",
                    "UTF-8"))
#Set variable with the checklists count, to be used in items and sections data loop
total_items <- Get_Checklists_Data$meta$page$total
```

Get checklist instances id rows:

Here we have two threaded loops. The first loop selects data from each checklist and the second loop selects each row within a given checklist, which may contain section or item data.

```
#Loop in checklists instances
for (i in 1:total_items) {

  pageoffset <- i-1

  Get_Checklists_URL <-
    paste(
      "https://developer.api.autodesk.com/bim360/checklists/v1/containers/",
      BIM360Container_Id[cont],
      "/instances?page[offset]=",
      pageoffset,"&page[limit]=1",
      sep = ""
    )

  Get_Checklists_Data <-
    jsonlite::fromJSON(content(
      Get_Checklists_Request,
      "text",
      "application/json",
      "UTF-8"
    ))

  Checklists_Content <-
    flatten(data.frame(Get_Checklists_Data["data"]))

  Checklists_Content_selected <-
    as.data.frame(Checklists_Content[, c(
      "data.relationships.container.data.id",
      "data.id",
      "data.attributes.scheduledDate",
      "data.attributes.location",
      "data.attributes.title",
      "data.attributes.createdAt",
      "data.attributes.createdBy",
      "data.attributes.completedItemsCount",
      "data.attributes.progress",
```

```
      "data.attributes.status.name",
      "data.relationships.createdFrom.data.id"
    )])

  # Insert checklist instance data in its data frame
  BIM360Checklists <-
    rbind(BIM360Checklists, Checklists_Content_selected)

  instance_id <- BIM360Checklists$data.id[i]

  Get_Checklists_instances_URL <-
    paste(
      "https://developer.api.autodesk.com/bim360/checklists/v1/containers/",
      BIM360Container_Id[cont],
      "/instances/",
      instance_id,
      "?include=sections.items.attachments",
      sep = ""
    )
       Get_Checklists_instances_Request <-
           GET(Get_Checklists_instances_URL, add_headers("Authorization" = Access_Token))

  Get_Checklists_instances_Data <-
    jsonlite::fromJSON(content(
      Get_Checklists_instances_Request,
      "text",
      "application/json",
      "UTF-8"
    ))

  Checklists_instances_Content <-
    flatten(data.frame(Get_Checklists_instances_Data["included"]))

  Checklists_instances_Content$instance_id <-
    rep(instance_id, nrow(Checklists_instances_Content))

  Checklists_instances_Content$included.attributes.answer <-
as.character(Checklists_instances_Content$included.attributes.answer)

  Checklists_instances_Content_selected <-
    as.data.frame(Checklists_instances_Content[, c(
      "instance_id",
      "included.id",
      "included.attributes.modifiedBy",
      "included.attributes.updatedAt",
      "included.attributes.status",
      "included.attributes.title",
      "included.attributes.number",
```

```
      "included.attributes.answer",
      "included.attributes.isRequired",
      "included.attributes.responseType.name"
    )])

    tot_inst_selec <- nrow(Checklists_instances_Content)


    #Loop in each checklists instances rows
    for (j in 1:tot_inst_selec) {

      if(Checklists_instances_Content[j,]$included.type == "instance_items")
        {
            # Insert checklist instance item data in its data frame
           BIM360Checklists_included_items <- rbind(BIM360Checklists_included_items,
Checklists_instances_Content_selected[j,])
        }

      if(Checklists_instances_Content[j,]$included.type == "instance_sections")
        {

          included_sections <-
flatten(data.frame(Get_Checklists_instances_Data$included$relationships$items$data[j]))

          if(nrow(included_sections)>0)
          {
              included_sections_selected <-
              as.data.frame(included_sections[, c(
                "id",
                "instance_id",
                "included.attributes.status",
                "included.attributes.title",
                "included.attributes.number",
                "data.attributes.instructions"
              )])

             # Insert checklist instance section data in its data frame
            BIM360Checklists_included_sections <- rbind(BIM360Checklists_included_sections,
included_sections_selected)
          }
        }
    }
}
```

Export data frames to ".csv" format

This final step saves the files in the ".csv" format, which are later imported into *Power BI*.

```
#Export in ".csv" format
write.table(BIM360Checklists,
        file = paste(dir_base,'/checklists/Checklists',cont,'.csv',sep=""),
        quote = FALSE,
        sep = "\t",
        row.names = TRUE,
        col.names = NA)

write.table(BIM360Checklists_included_items,
        file =
paste(dir_base,'/checklists_included_items/Checklists_included_items',cont,'.csv',sep=""),
        quote = FALSE,
        sep = "\t",
        row.names = TRUE,
        col.names = NA)

write.table(BIM360Checklists_included_sections,
        file =
paste(dir_base,'/checklists_included_sections/Checklists_included_sections',cont,'.csv',sep=""),
        quote = FALSE,
        sep = "\t",
        row.names = TRUE,
        col.names = NA)
```

Fully Automated Data Extraction

For the ultimate data extraction solution, we now work with the Python language, which extracts information from BIM 360®, handles data, and updates tables in a SQL Server database.

In *Python* it was possible to automate *Forge 3-legged OAuth flow*, using a BIM 360® user dedicated to extractions, with permission to view all data in the Projects.

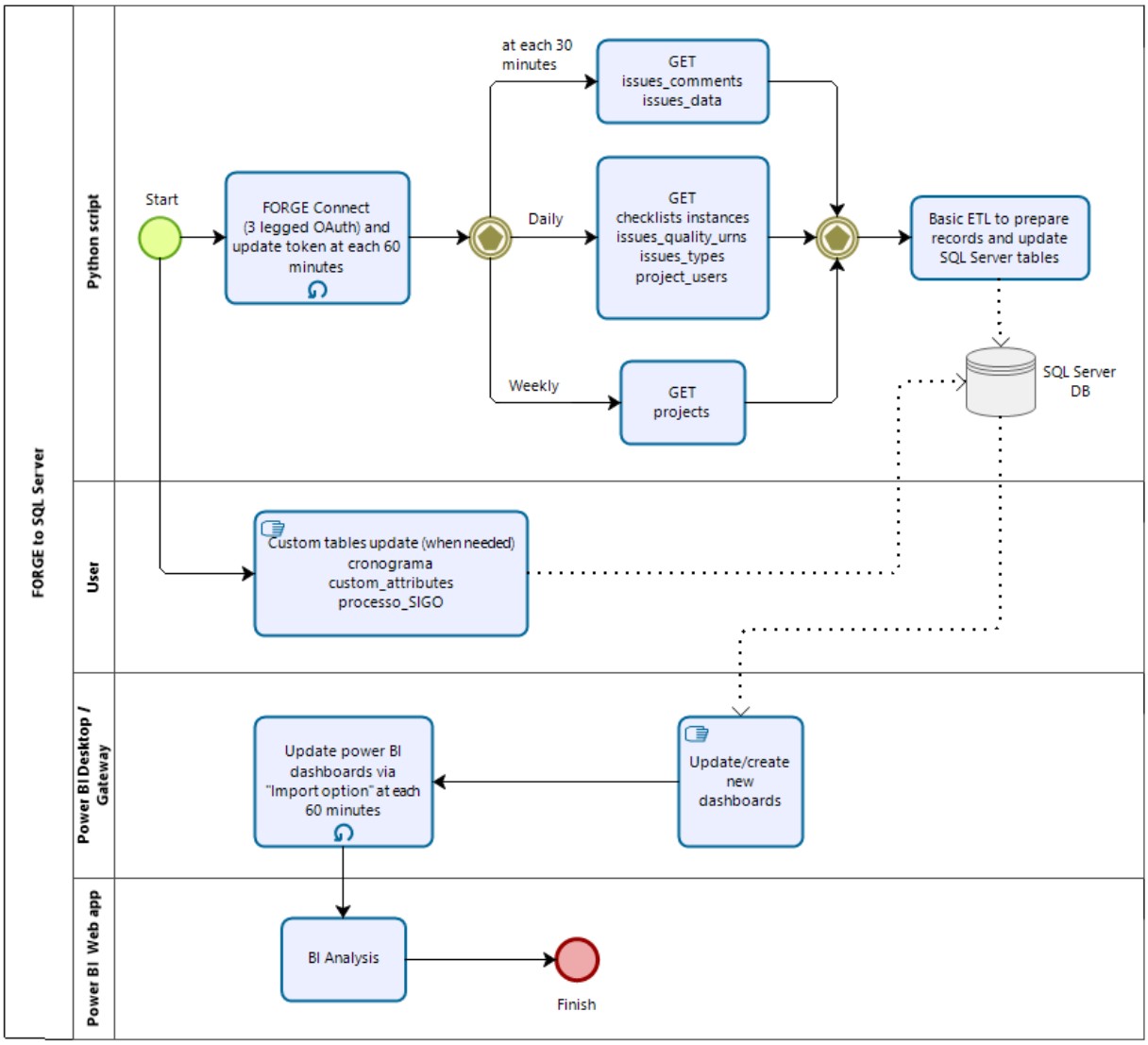
The policy for periodic database tables update has been set, as shown below:

Table	Refresh	Comments
issues_comments	At each 30 minutes	Optimize performance by loading only modified records after the last load (attributes/updated_at to filter data).
issues_data	At each 30 minutes	Optimize performance by loading only modified records after the last load (attributes/updated_at to filter data).
issues_custom_attributes	Query	Obtained from query in issues_data table
checklists	Daily – after midnight	
checklists_included_items	Daily – after midnight	
checklists_included_sections	Daily – after midnight	
issues_quality_urns	Daily – after midnight	
issues_types	Daily – after midnight	
project_users	Daily – after midnight	
projects	Weekly – on Sundays	
cronograma	Manual update	From Primavera P6 as report, planned future automated data extraction.
custom_attributes	Manual update	Data not yet accessible via FORGE. Future implementation.
processo_SIGO	Manual update	

Once BIM 360® data was accessible in DB *SQL Server*, the last step of automation was to prepare the environment for publishing dashboards developed in *Power BI PRO*.

The *Power BI Gateway* tool automates the publishing of updated dashboards, which is done every 60 minutes using the “Import” database access type.

Below is the workflow for automated BIM 360® data extraction:



Business Intelligence Analysis with Power BI

According to Wikipedia, the *Power BI* tool is a business analytics service by Microsoft. It aims to provide interactive visualizations and business intelligence capabilities with an interface simple enough for end users to create their own reports and dashboards.

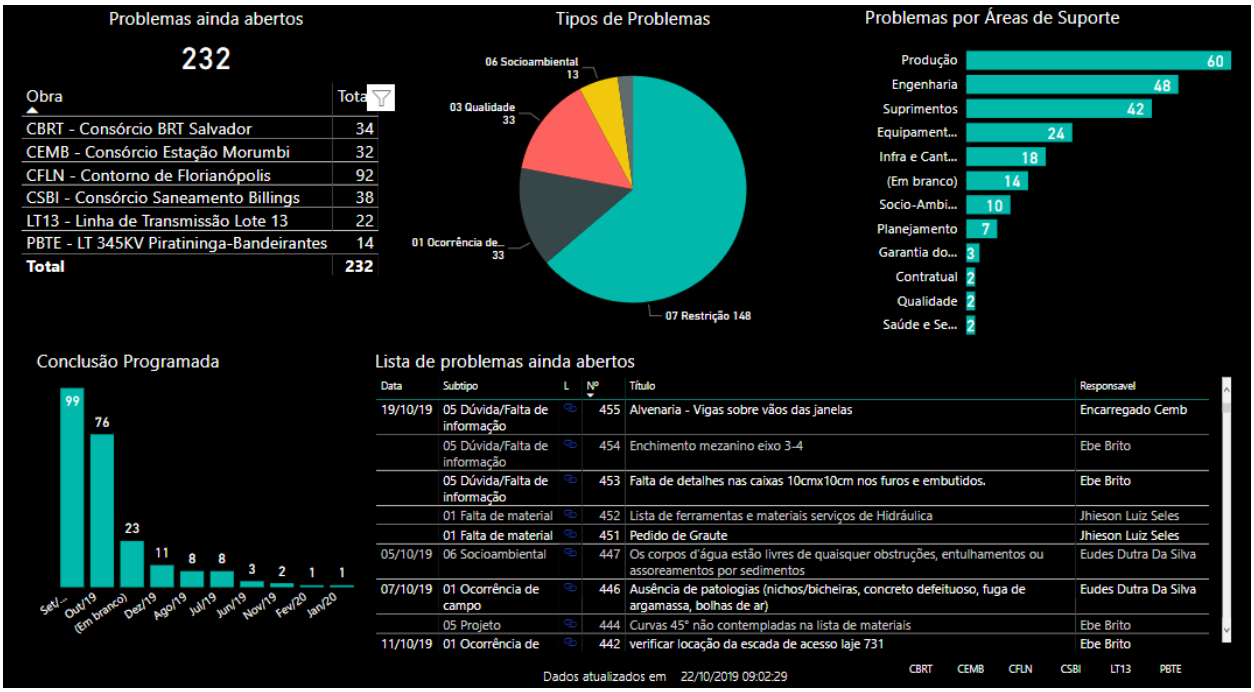
Power BI provides cloud-based BI services, known as "Power BI Services", along with a desktop-based interface, called "Power BI Desktop". It offers data warehouse capabilities including data preparation, data discovery and interactive dashboards.

Our BI environment allows access to BIM 360® information for all Company Projects, with the ability to access issue records or checklists directly inside BIM 360® from dashboard links.

Because BIM 360® data is available in a SQL Server database, analysts can create their own dashboards and perform ad-hoc analysis in *Power BI* or excel spreadsheets.

Two dashboards were developed that are available for all projects and are used in the routine Project follow-up meetings. These same dashboards are also accessible to functional managers, responsible for business processes, and can be consulted on touchscreen panels at process follow-up meetings.

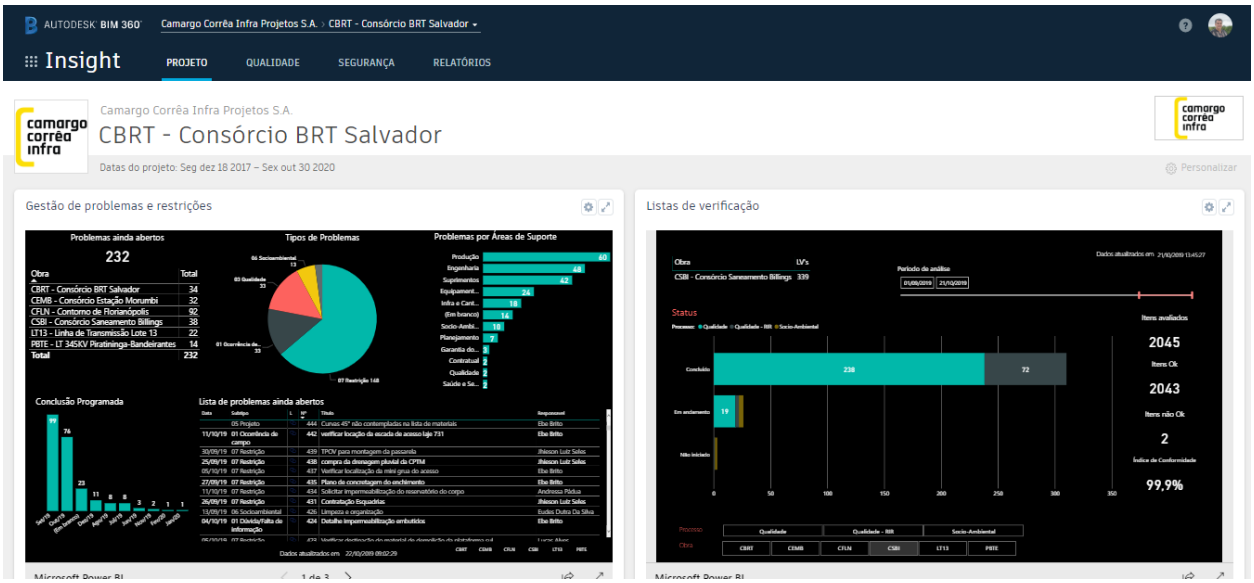
Dashboard 1: Issues management



Dashboard 2: Checklists



These panels have been incorporated into the BIM 360® Insight module (as *Power BI* object):



Current Limitations

This section describes the biggest limitations we faced so far on this implementation related to Forge and the customization.

BIM 360® API

- There's no "POST checklists" endpoint – This was needed for the programmatic upload of legacy checklists that had already been produced on projects that had already started before BIM 360® implementation started
- GET issues/:id doesn't return custom attributes – Despite being able to create and fill in custom attributes for Issues on BIM 360®, Forge doesn't handle them, hindering the use of this valuable resource.

Data Management API

- GET projects/:project_id/items/:item_id does not return the full metadata (custom attributes) from a document – BIM 360® does not create a full document list of a project, it only creates them folder by folder. This could be worked around with Forge, however, very little information is present on the response, making it impractical to create a project document list.

Further Development

The next steps for the development are:

- Get more information from BIM 360®, specially attachments from issues (photos);
- Building a project document list with full metadata through other means, whilst Forge isn't capable;
- Extract signature information from checklists, in order to rebuild checklist information into the client's template for handover.