Intro to Web Apis

Autodesk University 2019
SD322307-L
Murano 3205, Level 3, Lab 3
Tuesday, November 19 8:30am

Gui Talarico
@gtalarico

Learning Objects

- 💡 Discover basic concepts of Web APIs
- 🛠️ Learn how to use tools like Postman and Curl
- 🔒 Get a basic understanding of API Authentication
- 🧪 Experiment with Airtable and Forge and API

Description

Web APIs are the backbone of today's internet and are the most common way we connect to web-hosted platforms. By the end of this session, attendees will be familiar with the concepts and tools needed to interact with a public API, as well as have a basic understanding of how APIs work under the hood. This workshop will cover concepts such as HTTP protocol, Authentication Schemes, and API Specs. We'll also go through a few hands-on exercises and demonstrations of how to understand and interact with Forge and other APIs, while also learning useful tools like Postman, Curl, and Chrome Dev Tools.

Speaker

Gui Talarico is a Software Engineer at Airbnb. Prior to joining Airbnb, he was a Software Engineer at WeWork where his work focused on the integration of software development, physical space data, and design processes.

Beyond his professional work, Gui is an active contributor to open source projects, and online communities, including is Revit API Docs, an online Documentation platform for the Revit API. Since launching in late 2016, RevitApiDocs.com has received over 5 million views and has been widely recognized as a valuable resource to the Revit developer Community.

At Autodesk University 2017, Gui Talarico received the Best New Speaker Award as well as a Top Rated Speaker Honorable mention for his Hands on Lab session Untangling Python.

Links

- Github Repo
- Presentation
- Live Chat

Assistants

- Timon Hazell
- Pablo Gancharov
- ?

Software Requirements

- [ ] Chrome Browser
- [ ] Postman - Datasets/SF322307-L/portables/postman
- [ ] Cmder - Datasets/SF322307-L/portables/cmder

Terminology

API

Application Programming Interface is a form of a *contract* an application may offer as a way of providing functionality to other applications and services.

Client

An application that accesses resources and services made available by servers.

Server

An application that performs functionality and or serves resources to clients or other servers.

Web API

An API that provides functionality by means of web requests over the internet, most commonly through HTTP messages. Web APIs can be for internal, private, or public consumption. The

expected request and response formats define a contract between the client and server and are expected to be documented for other developers to use.

HTTP Protocol

HTTP is a protocol which allows the fetching of resources [...]
It is the foundation of any data exchange on the Web and a client-server protocol"
https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

Web Resource

An identifiable, "servable thing", usually identified by a URL (Uniform Resource Locator).

Common web resources are html pages, media (images, videos), files (zip, exe), JSON and XML objects, etc.

HTTP Protocol

HTTP or Hypertext Transfer Protocol is a protocol defined by Internet Engineering Task Force (IETF) and the World Wide Web Consortium. There are multiple versions and revisions of the protocol, but we will focus on HTTP 1.1 which was published in 1999.

The specification document is openly available and although technical, it's actually a great resource - RFC 2616

HTTP 1.1 x HTTP 2.0

2.0 is the most recent protocol and was published in 2014. Its key advantage is that it allows a client to make multiple requests using a single connection, where 1.1 requires each request to establish a new connection. According to W3Techs, as of September 2018, 29.7% of the top 10 million websites supported HTTP/2.

For this session, we will focus on the use of the 1.1 Protocol

Connection, Request, and Response Flow

When the client wants to communicate with a server, either being the final server or an intermediate proxy, it performs the following steps:

1. **Open a TCP connection**: The TCP connection will be used to send requests and receive responses. The client may open a new connection, reuse an existing connection, or open several TCP connections to the servers.
2. **Send an HTTP message**: HTTP messages (before HTTP/2) are human-readable. With HTTP/2, these simple messages are encapsulated in frames, making them impossible to read directly, but the principle remains the same.
3. **Read the response** sent by the server:
4. **Close or reuse the connection** for further requests.

(from https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview)

Request

```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept: *
```

Response

```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<html><body></body></html>
```

Request & Response Structure

Request

```
{ METHOD } / HTTP/1.1
Host: developer.mozilla.org
{ OPTIONAL: REQUEST HEADERS }

{ OPTIONAL - REQUEST BODY }
```

Response

```
HTTP/1.1 { STATUS CODE }
{ HEADERS }
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

{ BODY }
<html><body></body></html>
```

HTTP Methods

GET

Requests a resource. Requests using GET should only retrieve data.

POST

Sends data to the server. Post requests are usually used to create a new resource or save data on a web server. These request normally include content (aka body, or payload).

The type of the body of the request is indicated by the Content-Type header.

PUT

Creates or replaces a request.

"The difference between PUT and POST is that PUT is idempotent: calling it once or several times successively has the same effect (that is no side effect), where successive identical POST may have additional effects, like passing an order several times."

DELETE

The HTTP DELETE request method deletes the specified resource."

PATCH

"The HTTP PUT method only allows complete replacement of a document. Unlike PUT, PATCH is not idempotent, meaning successive identical patch requests may have different effects. However, it is possible to issue PATCH requests in such a way as to be idempotent.

PATCH (like PUT) may have side-effects on other resources."

Others

Other methods less commonly used include CONNECT, HEAD, OPTIONS, TRACE.

Status Codes

"HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes: informational responses [100s], successful responses [200s], redirects [300s], client errors [400s], and servers errors [500s]. Status codes are defined by section 10 of RFC 2616."

1xx

Informational - not commonly used.

2xx

Successful responses.

200 OK

Your request went through and a response was returned.

201 CREATED

Your request went through and a new resource was created. Response generally includes an id or some other mean of locating the resource.

3xx

Redirect responses.

301 MOVED

The resource has permanently moved. The response should include the new location.

4xx

Error Response.

400 BAD REQUEST

The server could not understand the request. This could be due to a poorly formatted request, invalid syntax, or missing data.

401 UNAUTHORIZED

Authentication is missing or incorrect.

403 FORBIDDEN

The server understood your request and know who your are, but has decided you should not be served the requested resource.

404 NOT FOUND

The server could not find the requested resource

405 METHOD NOT ALLOWED

The server does not support the provided method for the selected resource.

418 I AM A TEAPOT

I am not kidding. Look it up

5xx

The server has encountered an error and is unable to fulfill the request.

The response should include an explanation or details about the error.

500 INTERNAL SERVER ERROR

"The server encountered an unexpected condition which prevented it from fulfilling the request."

This usually means an exception has occurred. The cause might be a bug in the server application, or because you provided an unexpected request payload and exception was raised because of it.

502 BAD GATEWAY

"The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request."

Authentication Schemas

HTTP Protocol provides a variety of means of authenticating requests. These allow a server to verify identity and permissions before serving a resource.

Http schema is defined in the request headers using the format
`“Authorization: <schema> <auth-data>”`

Basic Auth

Basic auth expect the client’s credentials to be encoded using Base64 encoding.

The way you create the request depends on what type of tool you are using to prepare the request (see Tooling section below). In python, it might look something like this:

```
>>> credentials = b"username:password"
>>> b64credentials = base64.b64encode(credentials)
>>> print(b64credentials)
b'dXNlcm5hbWU6cGFzc3dvcmQ='
```

Once you have the encoded credentials, the final HTTP request would be:

```
GET / HTTP/1.1
Host: secure.website.com
Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=
```

Some clients accept the credentials to be provided directly on the URL, although this is often considered bad practice. A request to `username:password@secure.website.com` would be equivalent to the request described above

Bearer

Bearer schema allows the client to access a protected resource using an access token instead of using credentials, as defined in the Oauth 2.0 framework (RFC 6749).

The provided access token can then be reused by the client for subsequent requests until the token revoked or expires.

A simple bearer type authentication is used by the Airtable API, which we will use later in the workshop. From your account settings page, you are able to request an access token, which can be used to access your Airtable data. A request would look something like this:

```
GET /v0/basekey/basname HTTP/1.1
Host: api.airtable.com
Authorization: Bearer {YOUR_AIRTABLE_API_KEY}
```

Oauth 2.0

Oauth 2.0 is an access delegation standard, allowing the client to access server resources without exposing credentials.

Oauth 2.0 was published in 2012, also by IETF and is the recommended (often required) authentication mechanism of all major technology platforms including Facebook, Google, Twitter, Github, etc.

Autodesk Forge also uses Oauth 2.0 and provides 2 mechanisms for accessing data, 2-legged and 3-legged. We will use these flows to explain how the Oauth protocol can be used.

Oauth 2-legged

Forge uses 2-legged authentication as a simple way of giving access to data for a Forge App.

In this scenario, we will use credentials issued to a Forge App to get an access token, and then use this token to obtain data belonging to this app.

Further details are outlined here:

https://forge.autodesk.com/en/docs/oauth/v2/tutorials/get-2-legged-token/

1. Create a Forge App
2. Obtain a **client_id** and **client_secret**
3. Make a request to Forge's authentication endpoint to obtain an **access_token**

Request

```
POST /authentication/v1/authenticate HTTP/1.1
Host: developer.api.autodesk.com
Content-Type: application/x-www-form-urlencoded

client_id=obQDn8P0GanGFQha4ngKKVWcxwyvFAGE&
client_secret=eUruM8HRyc7BAQ1e&
grant_type=client_credentials&
scope=data:read
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
{... other headers ...}

{
```

```
  "token_type": "Bearer",
  "expires_in": 1799,
  "access_token": "eyJhbGc … 8B2nSjrq_ys"
}
```

With the access token in hand, you can now request secure resources from the server by including the header in your next request `Authentication: Bearer {access_token}`

Oauth 3-legged

This flow is used when a server wants to obtain access rights on behalf of a user.

Unlike 2-legged, in which authentication is performed between a client application and a server (2 parties), 3 legged also includes a user.

This flow is seen often when you want to authorize an application to access another service on your behalf (let an app post using your Twitter account; let a CI server access your private Github repository, etc).

We will once again use Forge as an example. Details are outlined here:

https://forge.autodesk.com/en/docs/oauth/v2/tutorials/get-3-legged-token/

1. The client sends the user to an authentication prompt hosted by the service provider
2. The server will authenticate the user and ping back the client app at an agreed “callback” url with an authentication **code**
3. Client App calls the server with **client_id** and **client_secret** but also the user’s authorization **code.**
4. The server will respond with an **access_token**.
5. This access token can be used by the application to make requests on behalf of the user

REST

REST was defined in 2000 as a result of doctoral dissertation, “HTTP object model”. The paper defined an “architectural style” that allowed developers to create consistent stateless abstraction using HTTP methods (verbs), and URLs (Uniform Resource Locator). An API that follows “REST style” is said to be RESTful.

Rest Style Calls

URL Endpoint	GET	PUT	PATCH	POST	DELETE
api.com/resources/	Lists all resources	Replace the entire collection with another collection.	Not generally used	Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.	Delete the entire collection.
api.com/resources/item1	Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	Replace the addressed member of the collection, or if it does not exist, create it.	Update the addressed member of the collection.	Not generally used. Treat the addressed member as a collection in its own right and create a new entry within it. [17]	Delete the addressed member of the collection.

Note: The original REST paper did not include a technical implementation, so there are differing interpretations of its proper usage.

Cookies

Http is by definition a **Stateless Protocol**: state is not maintained between requests. To understand what stateless means in this context, imagine the following scenario:

A person enters a Bank branch and shows their Id, debit card, and pin. At this point, the client is “authenticated” - and any services or resources requested will be provided without further authentication details. Give me my balance. Give me $50. Deposit this check. Etc. Once authenticated, “state” is maintained across the requests (bank knows client id, and the client is authenticated).

With HTTP, since state is not guaranteed, the user would have to authenticate again at every request. Cookies provide a way for the server to ask the client to store information and resend it on every subsequent request, allowing a “state” to be maintained.

In the bank scenario, after showing the id, debit card, and pin, the bank would say: “Client, save this cookie: `SessionId: abc-client-session-123`, and next time you need something, give me this code and I will know it’s you.

In other words, cookies are used to maintain state and store information relevant to servers and clients across requests. The following requests explain its basic usage:

Request

```
GET /recipe/apple-pie HTTP/1.0
Host: www.food.com
Authentication: Basic as123asds==
```

Server Response

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: last-recipe-requested=apple-pie
Set-Cookie: session_id=zxczxcasd; Expires=Wed, 21 Oct 2015 07:28:00 GMT;
```

This will tell the client to store to store two cookies, and send them over in the following requests:

Following Requests

```
GET /recipe/hamburger HTTP/1.0
Host: www.food.com
Cookie: last-recipe-requested=apple-pie; session_id=zxczxcasd;
```

Tooling

cUrl

cUrl is a simple yet powerful command line client application for making HTTP requests (other protocols are also supported).

Curl is installed by default in MacOS and Linux. Windows users can manually install Curl for Windows or installed the amazing Cmder terminal, which includes cUrl and many other helpful command line applications. A simple curl HTTP request is structured as follows:

```
curl -X {METHOD} "{HOST}" -H "{HEADERS}"
```

Example:

```
curl -X GET "https://www.server.com/users/" -H "Accept: application/json"
```

This would prepare the HTTP request message and send it to the server. Add `-i` or `-v` to print additional details like request and response headers.

```
curl -X GET "https://www.server.com/users/" -H "Accept: application/json"

GET /users/ HTTP/1.0
Host: www.server.com
Accept: application/json

{
  "users" : [
     ...
  ]
}
```

Postman

Postman is a sleek and easy-to-use desktop client with a variety of tools for working with web requests.

In addition to preparing and issuing HTTP requests, it can also create and manage collections, generate documentation, and run tests.

Postman is widely used to test APIs, whether they are third party or one you are developing yourself.

Postman Documentation

Chrome Dev Tools

Chrome dev tools a suite of tools that are included with Google Chrome Browser.

Dev Tools is an invaluable tool for understanding and debugging API calls that are made by web applications.

You can access the tools by pressing going to Settings > More Tools > Developer tools.

The Network tab in the DevTools panel can be used to “inspect” requests on a given page to understand the request and response.

Hands On

Now that we have a built a basic vocabulary, let's dive in a learn how to explore and interact with 2 APIs: Airtable API, and Forge API

Airtable API

Airtable is one of the most flexible tools I know for creating quick relational datasets. It is the perfect combination between the flexibility of Google Sheet and the integrity of relational database, all with an intuitive and enjoyable interface.

You can use Airtable for variety of uses, including: task management, project management, and many more. Checkout these templates for more examples.

Conveniently, Airtable also has an API that can be used to read and create records. In the examples below, we will learn how explore the API and make some sample requests.

Create an Airtable Account

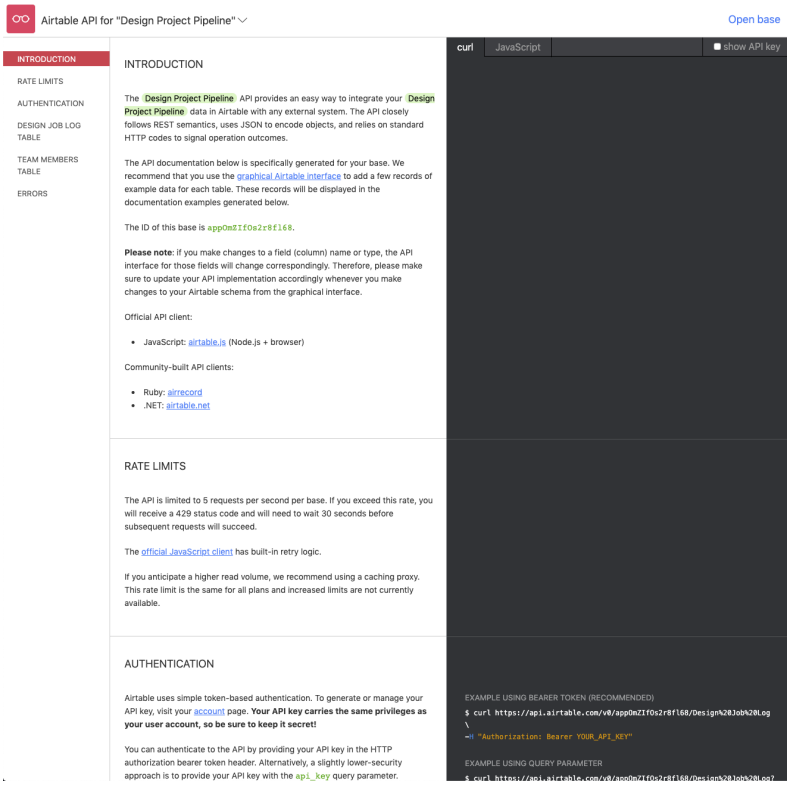
If you don't have an Airtable account, head over to airtable.com and create one. You can always use my referral link which gives me some free credits:
https://airtable.com/invite/r/JkBjLFMa

Once you have an account, go to airtable.com/api and open one of the Sample Tables, preferably the "Design Project Pipeline"

Reading the Docs

One of most critical parts of using an API, is ensuring you are familiar with it's documentation. The docs should tell you everything you need to know about the API, including how to authenticate with it, how to fetch, create, update, and delete data.

The Airtable docs it's particular good in that the documentation is dynamically generated and it uses the actual url endpoints and API key you will be interacting with.



Introduction

Note the introduction section includes the "Base Id" of the this base (airtable spreadsheet). This ID will be used in the URL path we will call.

Rate Limit

Apis usually have a Rate Limit to limit the rate at which a user can call it. This prevents accidental or malicious abuse of the API service. Imagine the compute resources you would consume on the server side if you were to call the API 1,000 in one second.

Authentication

The authentication session explains how you are expected to authenticate with it.

Airtable requires you to provide an API Key on each request, and it allows you to provide the request in two different ways: through the request Header, or through the URL. For this example we will use the header option.

We will try the authentication in the section below.

```
# Header Authentication
curl "https://api.airtable.com/v0/appOmZIfOs2r8fl68/Design%20Job%20Log"
     -H "Authorization: Bearer YOUR_API_KEY"

# Url Parameter Authentication
curl "https://api.airtable.com/v0/appOmZIfOs2r8fl68/Design%20Job%20Log?api_key=YOUR_API_KEY"
```

Overview

With authentication, out of the way, using the Api become fairly easy, especially given the quality of it's docs.

```
"https://api.airtable.com/v0/{BASE_ID}/{TABLE_NAME}?parameters"
     -H "Authorization: Bearer YOUR_API_KEY"
```

Table & Fields

This next section tells us about each of the tables on the "base", and what data types we expect.

List Records

Retrieves a list of records for a a table on a given view. Note how the documentation offers a varieties of "parameters" to configure the request - filter, sort, format, etc.

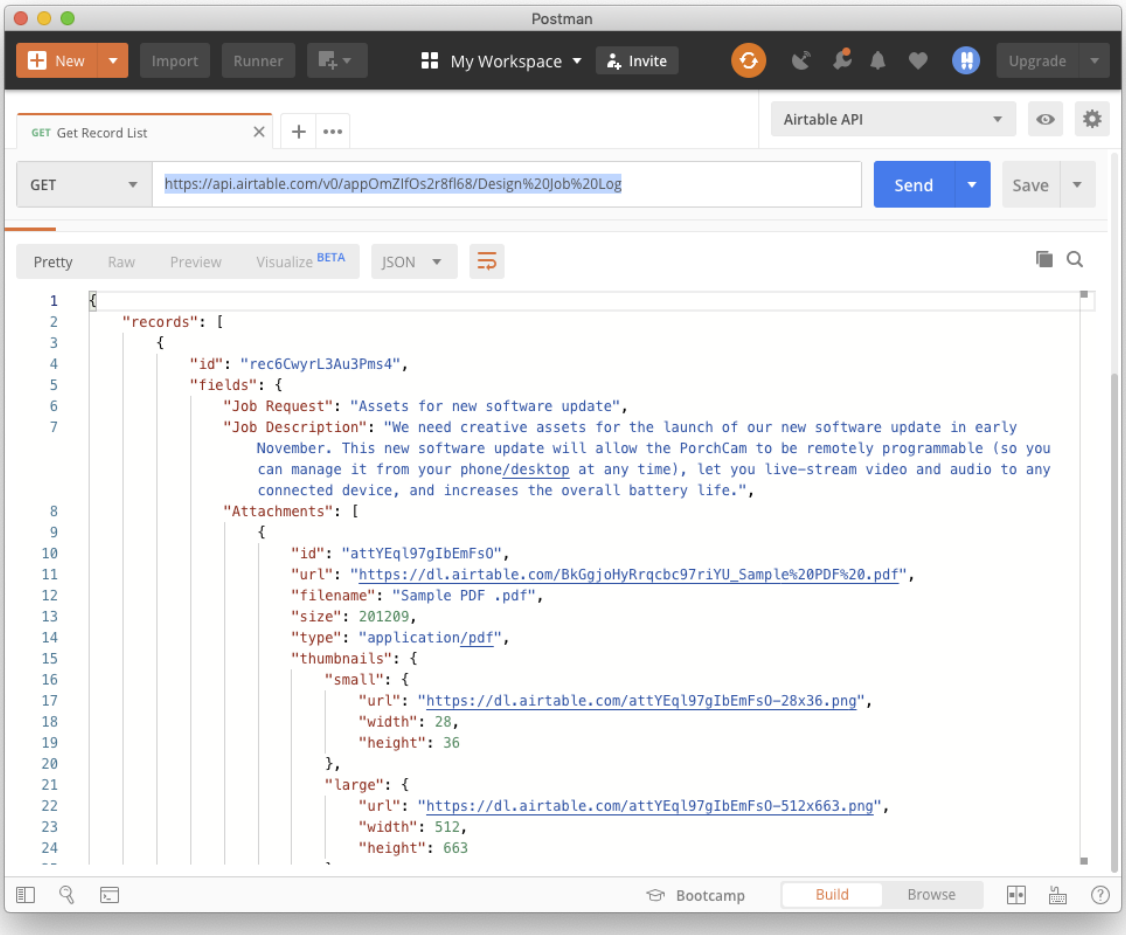
We will make two requests below, in cUrl and Postman:

Curl

```
curl "https://api.airtable.com/v0/appOmZIfOs2r8fl68/Design%20Job%20Log?maxRecords=5"
     -H "Authorization: Bearer YOUR_API_KEY"
```

Postman

- GET: https://api.airtable.com/v0/appOmZIfOs2r8fI68/Design Job Log
- Headers: `Authorization` : `Bearer {YOUR_API_KEY}`



Forge Platform API

Forge is a cloud-based platform by Autodesk. The platform is accessible through a collection of collection of APIs, and allow developers to build services leveraging Autodesk's Cloud tools and services. Some examples of tools accessible through Forge include:

- 2D and 3D Web Viewer
- Data Management API → Access data from BIM 360

- Model Derivative API → Convert design files into other formats
  eg. `.rvt` into `.stl` so it can be viewed through the Web Viewer
- Design Automation API → Execute automation tasks on your Revit files directly on the cloud

Given the length and scope of this session, we will not be able to go into great depth, however, the documentation is comprehensive and include easy to use sample projects.

On this hands-on exercise, we will try to get you comfortable with exploring and learning how to use the documentation, as well as do a few tests using Postman.

Prerequisites

- An Autodesk Account
- Postman

Creating an App

The content below includes references the documentation on the Get Started Docs

Sign in and head to **forge.autodesk.com/myapps**

1. Within the My Apps page, click **Create App**
2. Select all the services you want to use for your app (select all)
3. Fill in the mandatory fields in the App Information section
    1. App Name: `au-demo-yourname`
    2. App Description: `Test demo app`
    3. Callback URL: `http://notreallyneeded.com`
    4. Website Url: *leave it blank*

The Callback URL is used to authenticate a user using a 3-legged oauth flow. Since we are not building a full application we won't need this, but since the field is required just put any website url.

After creating the app you will be taken to your App's page, where you can obtain a Client Id and Client Secret. These two values are the credentials we will use to connect to Forge.

Authentication (Authentication API)

In this first session we will review the Authentication API and learn about the Authentication Flow. The documentation for this API is here forge.autodesk.com/en/docs/oauth/v2/developers_guide/basics/

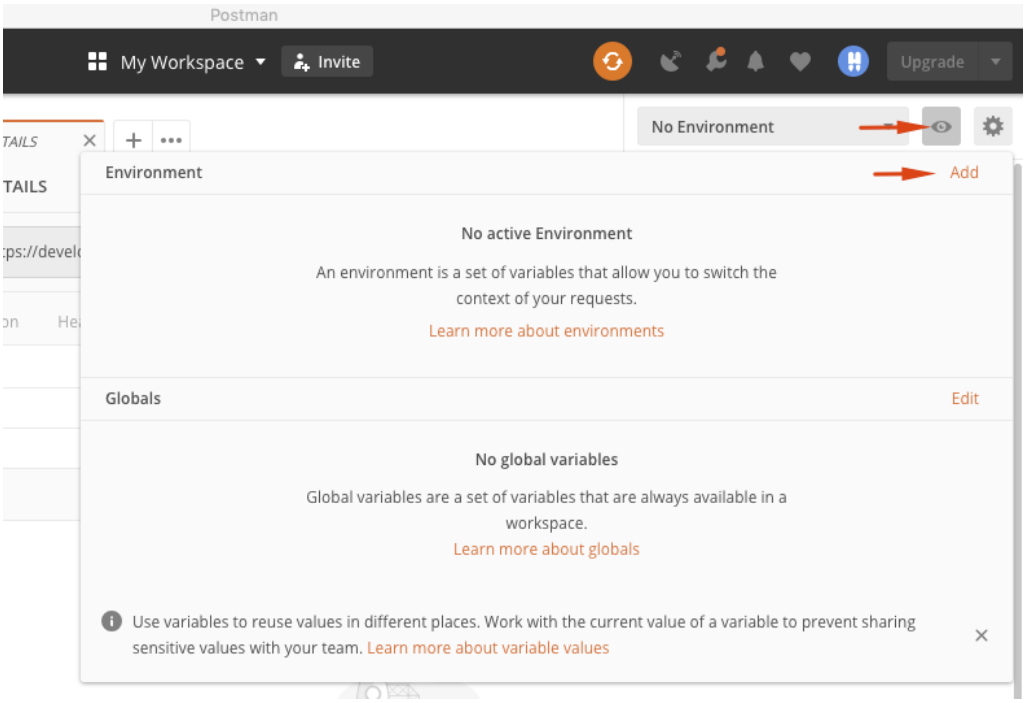
The most important concept of this session is to understand the different ways we authenticate with Forge. Let's review a few key terms:

- **Application** → the app we just created
- **User** → Some user with an Autodesk account
- **Forge API** → the service we request data from
- **2-Legged Authentication** → Authentication between **Application** and **Forge API.**
  This is a requirement, but it's also limited. An application has a limited set of actions it can perform on Forge - eg.an application doesn't own "models" and not part of Bim 360 teams.
- **3 Legged Authentication** → The **Application** authorize with **Forge API** on behalf of a **User** - ie. the grants the application authorization to request data on their behalf.

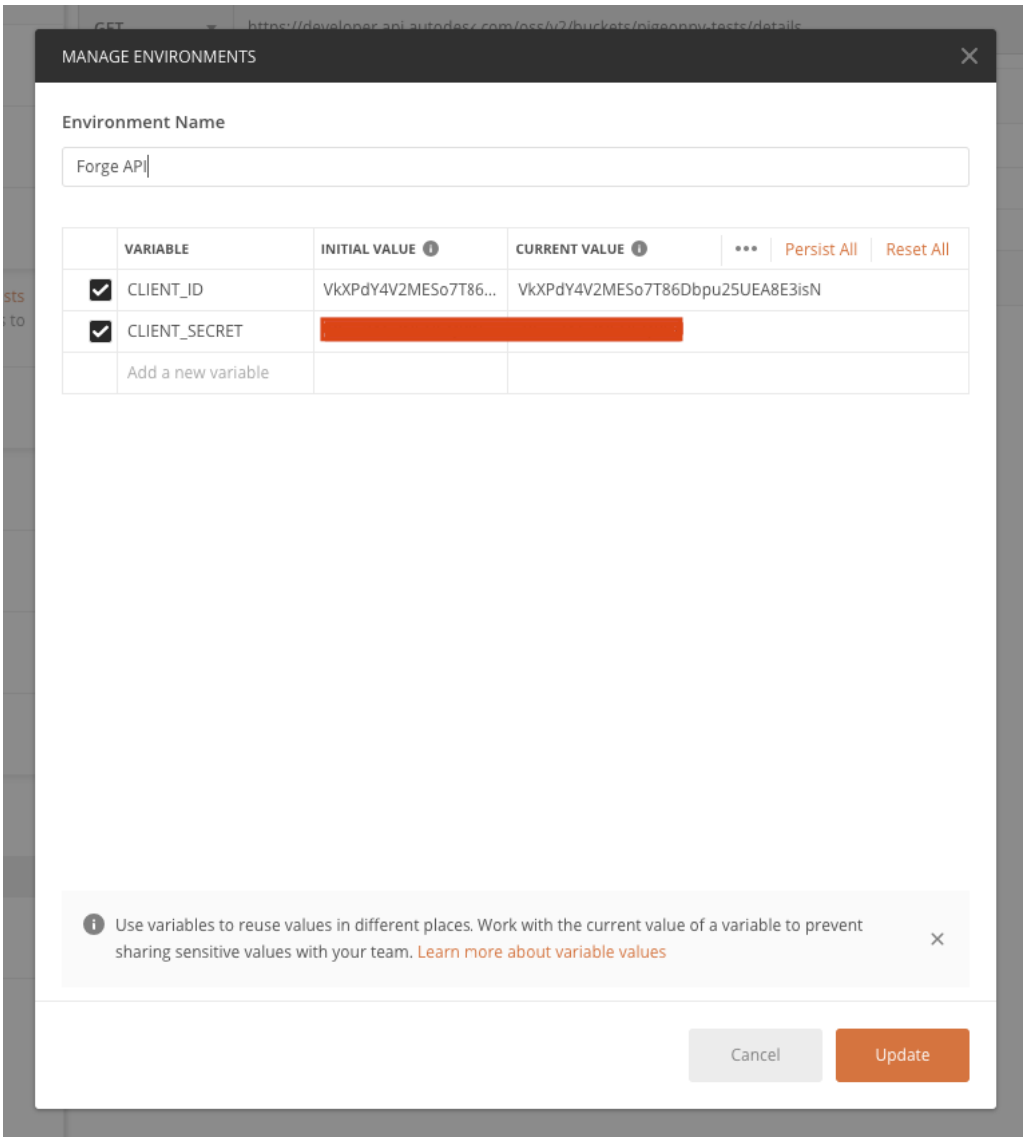
To perform most requests we are interested in we need to perform a 3-legged authentication, but first let's setup our Application credentials on postman and verify them using a 2-Legged authentication (Application → Forge API)

Create a New Environment in Postman

1. Click on the Environment Preview button (the "Eye")
2. Click **Add**
3. Add your Client Id and Client Secret value (`CLIENT_ID` and `CLIENT_SECRET`)
4. Click **Update** to confirm


Postman
My Workspace
Invite
Upgrade
No Environment
Environment
Add
No active Environment
An environment is a set of variables that allow you to switch the context of your requests.
Learn more about environments
Globals
Edit
No global variables
Global variables are a set of variables that are always available in a workspace.
Learn more about globals
Use variables to reuse values in different places. Work with the current value of a variable to prevent sharing sensitive values with your team. Learn more about variable values

Steps 1 and 2



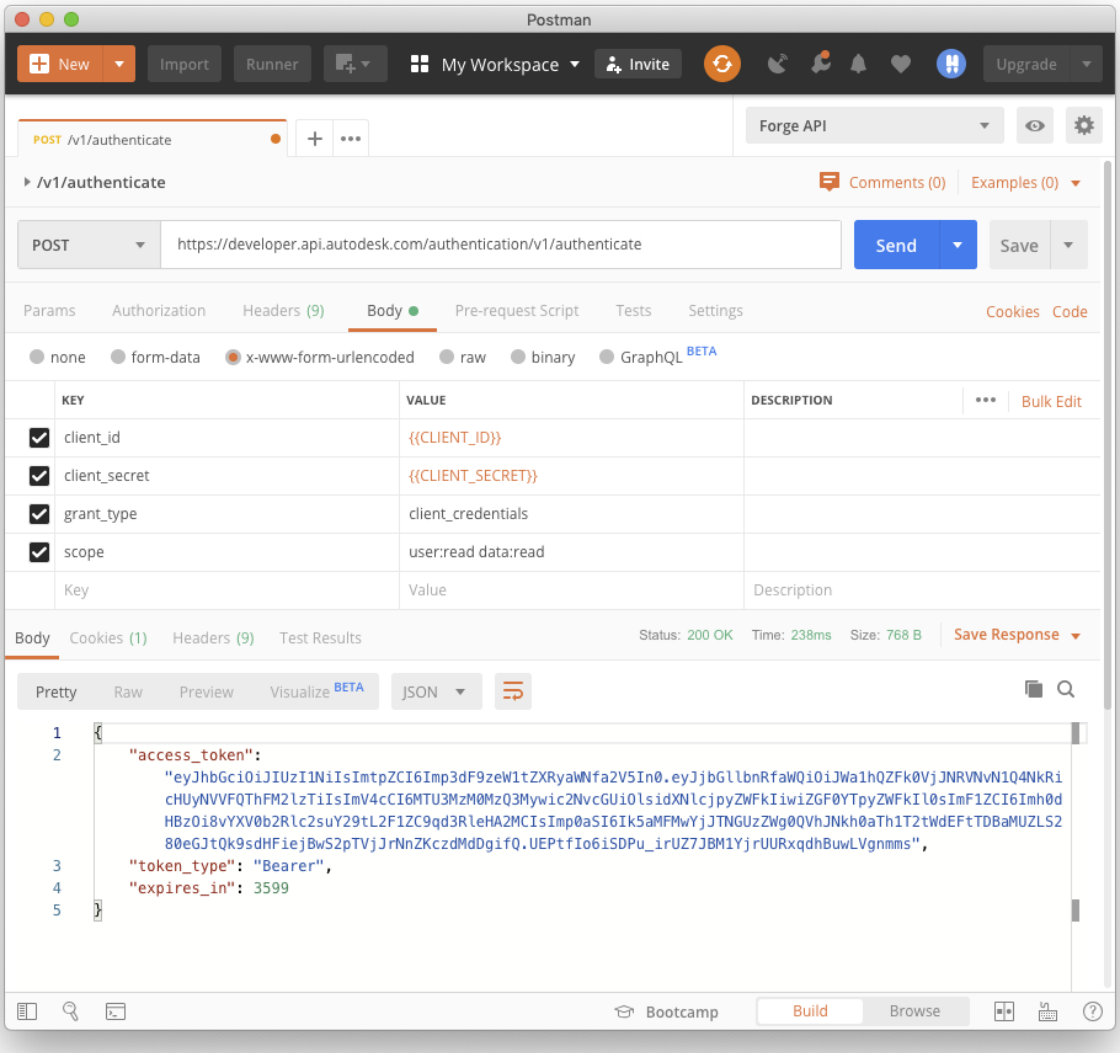
Steps 3 and 4

Verify Application Authentication

> The details of the request we will make are describe here: forge.autodesk.com/en/docs/oauth/v2/reference/http/authenticate-POST/

After reviewing the docs, we define all the components of the request:

- Method: `POST`
- Endpoint: `https://developer.api.autodesk.com/authentication/v1/authenticate`
- Headers:
  - `Content-Type: application/x-www-form-urlencoded`
- Body:
  - `client_id`
  - `client_secret`
  - `grant_type`
  - `scope`



 If you are feeling Nerdy you can test the same request using cUrl

```
curl -X POST \
  https://developer.api.autodesk.com/authentication/v1/authenticate \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  -d 'client_id={CLIENT_ID}&{CLLIENT_SECRET}&grant_type=client_credentials&scope=user%3Aread%20data%3Aread'
```

3-Legged Authentication

Now let's setup authentication for our application on behalf of a user. This makes the authentication process slightly more complicated but Postman makes it easy.

Let's review the steps of the 3-legged oauth:

- **User** makes a request to **Application**
- **Application** redirects user to an authentication flow managed by **Forge**
  This allows the user to login without exposing their credentials.
  A "callback" url is provided, so the flow can redirect the **User** back to the **Application**
- After successful login, Forge send the user to the provided callback address along with a "code". This "code" will allow the Application to make requests to Forge on their behalf
- Application makes a request to Forge, providing the Application credentials along with the "code" so it can act on behalf of the **User**

This article on AdnDevBlog includes a details step by step, but we will include the key steps here as well:

- Create a New Postman Request
- Under **Authorization** Tab, select Oauth2.0
- Click **Get New Access Token**

In this form, we will configure Postman to be able to complete the authentication flow on behalf of a User.

- **Token Name** → Any name for you to remember this Token Request

  `Forge Oauth Token`
- **Grant Type** → Describes the way in which Postman will attempt to authenticate with Forge. You can read more about the options here.

  `Authorization Code`
- **Auth URL** → The url the user should be sent to authenticate (the Login Form)

  `https://developer.api.autodesk.com/authentication/v1/authorize`
- **Access Token URL** → The url Postman will call to get the access token

  `https://developer.api.autodesk.com/authentication/v1/gettoken`
- **Client ID** → The Client Id for our App - from our Postman Environment

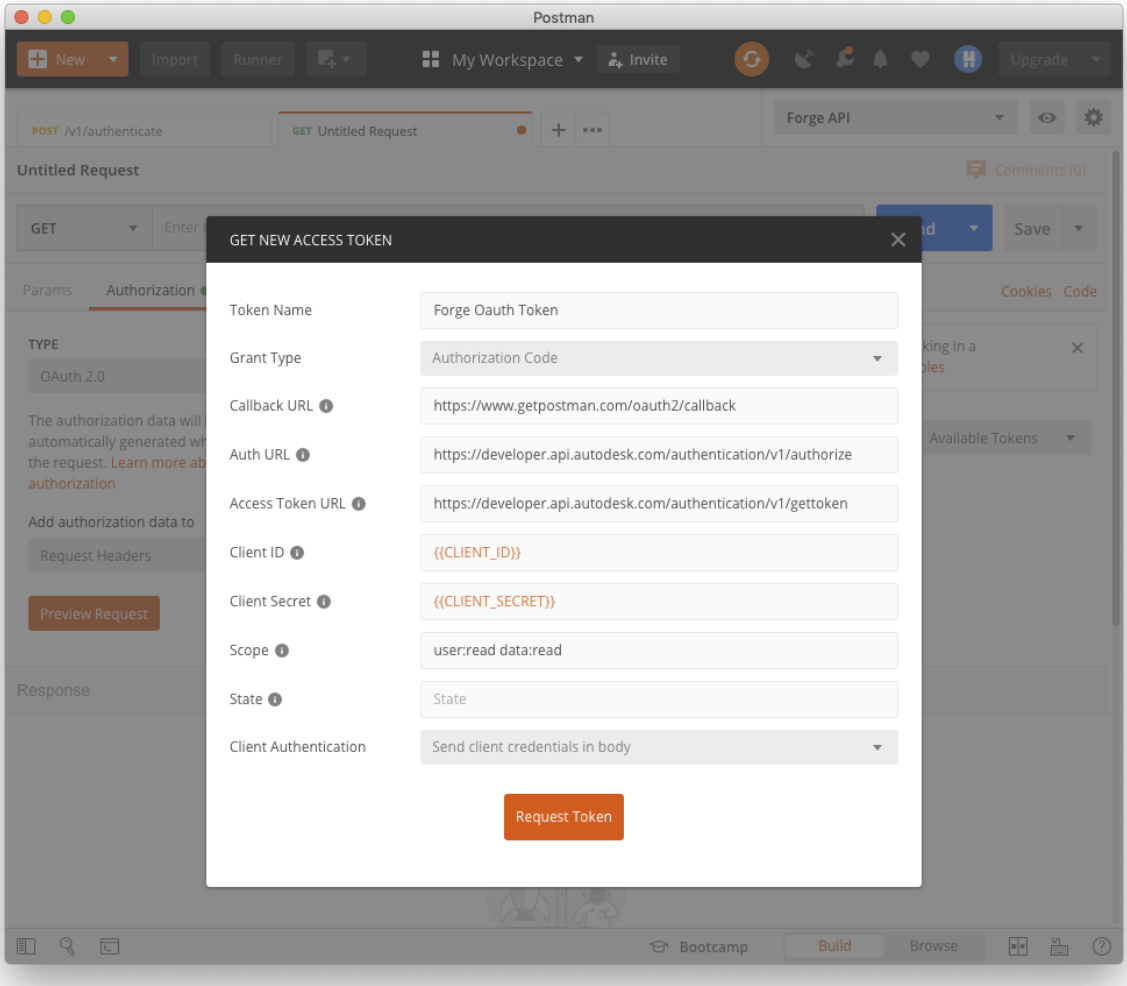
  `{{CLIENT_ID}}`
- **Client Secret** → The Client Id for our App

  `{{CLIENT_SECRET}}`
- **Scope** → The scope, or permissions the user wants to grant the app. A a full list of scopes is listed here. "Scopes" should be separated by a space. The scope below will let the application "read" user details and data on their behalf.

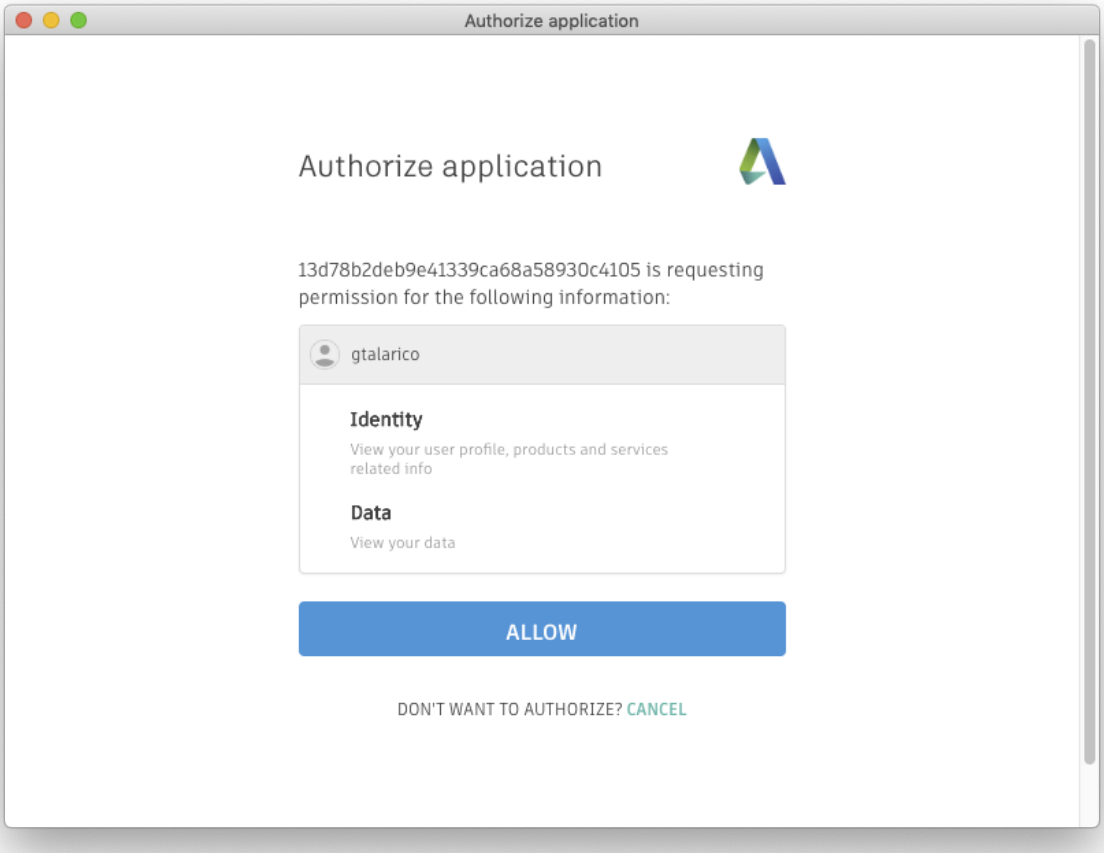
  `user:read data:read`
- **State** → Leave it blank.

The Flow is the following:

1. Configure the required parameters so Postman can perform a 3-Legged Authentication and get an access token
2. Login as a User
3. Stave the resulting Access Token

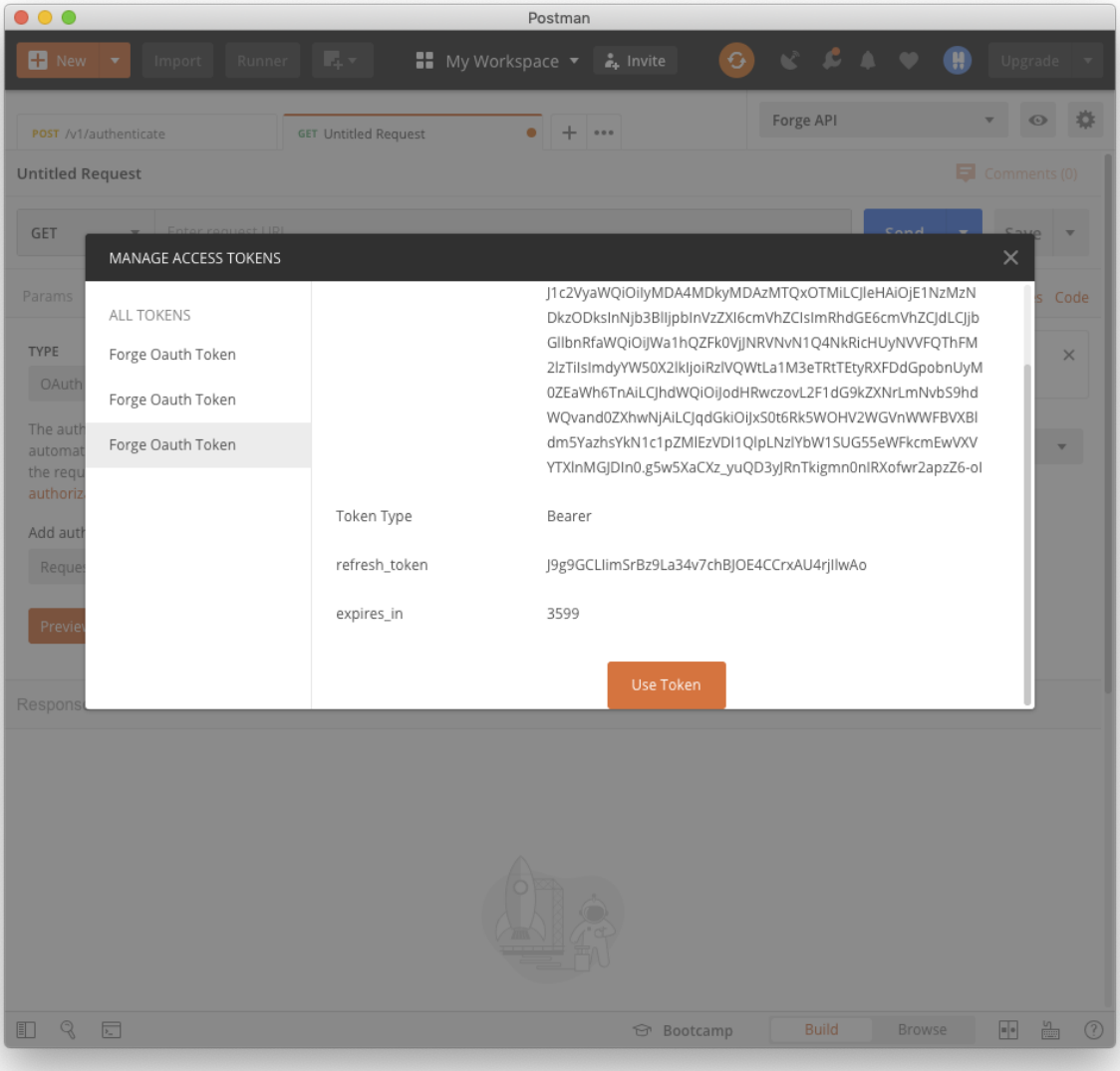

Postman
New
Import
Runner
My Workspace
Invite
Upgrade
POST /v1/authenticate
GET Untitled Request
Forge API
Untitled Request
Comments (0)
GET
Save
Params
Authorization
Cookies
Code
TYPE
OAuth 2.0
Add authorization data to
Request Headers
Preview Request
Response
Available Tokens
GET NEW ACCESS TOKEN
Token Name
Forge Oauth Token
Grant Type
Authorization Code
Callback URL
https://www.getpostman.com/oauth2/callback
Auth URL
https://developer.api.autodesk.com/authentication/v1/authorize
Access Token URL
https://developer.api.autodesk.com/authentication/v1/gettoken
Client ID
{{CLIENT_ID}}
Client Secret
{{CLIENT_SECRET}}
Scope
user:read data:read
State
State
Client Authentication
Send client credentials in body
Request Token
Bootcamp
Build
Browse

Step 1



Step 2

⚠️ After Token is acquired in the step below, be sure to scroll down and click the button **Use Token**



Step 3

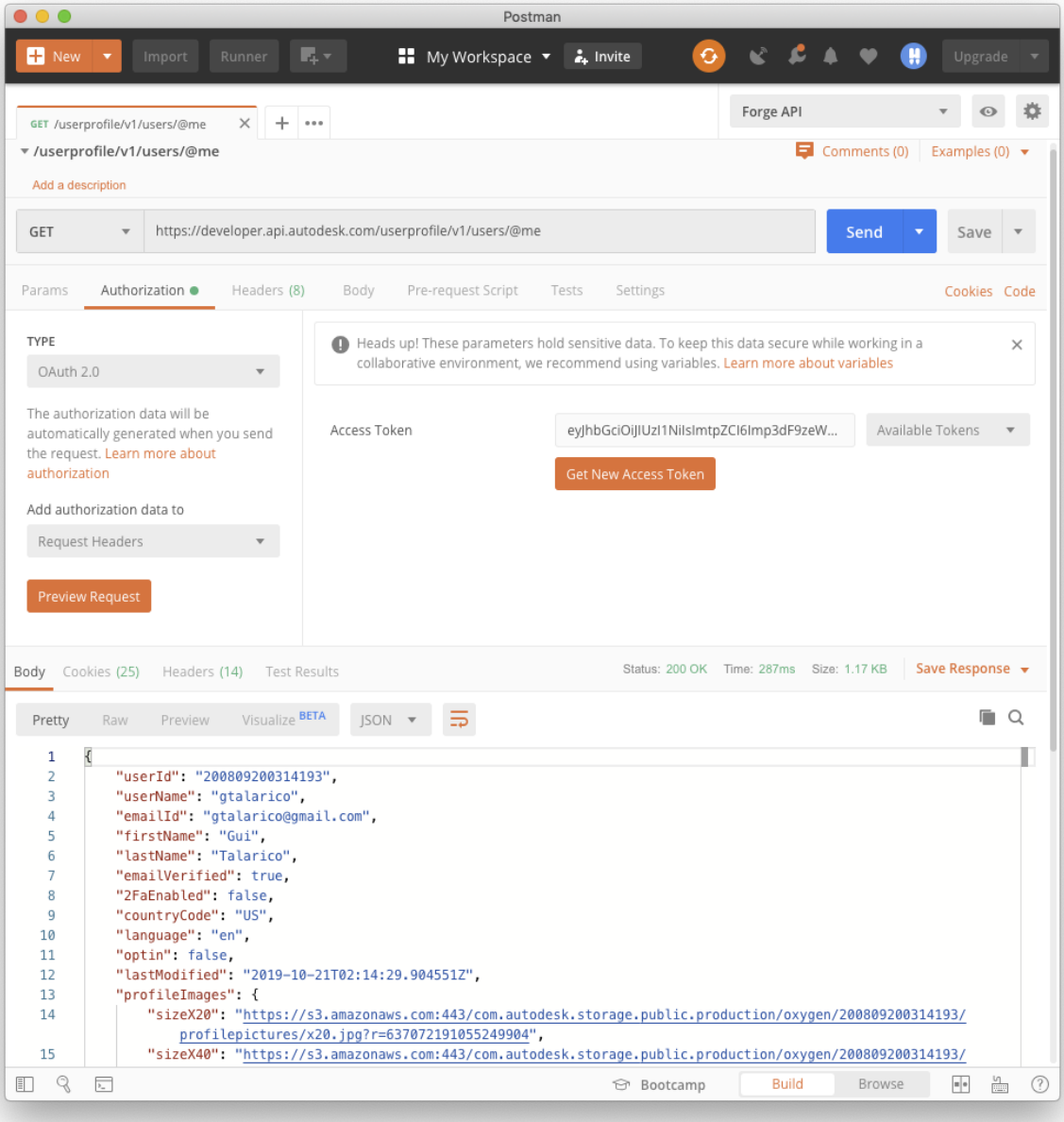
Verify Access Token

We will now verify the Token works by fetching our User's Profile, as documented here: forge.autodesk.com/en/docs/oauth/v2/reference/http/users-@me-GET

After reviewing the docs, we define all the components of the request:

- Method: `GET`
- Endpoint: `https://developer.api.autodesk.com/userprofile/v1/users/@me`
- Headers (automatically added by Postman)

- `Authorization: Bearer eyJhbGciOiJIUzI....`



Using Other Endpoints

Now that we have setup authentication, let's head of to the Data Management API and try fetching some data.

After reviewing the API Reference, we decide we ant to get a list of Projects this User has access to. Since Projects are part of a hub, we need to first get a list of Hubs the user belongs to:

- GET hubs

  Get a List of "hubs" the user belongs to.

- GET hubs/:hub_id/projects

  Using one of the hub id's from the previous request, we will now fetch all of the projects in the given hub.

- GET hubs/:hub_id/projects/:project_id/topFolders

  Get the folders in the a project

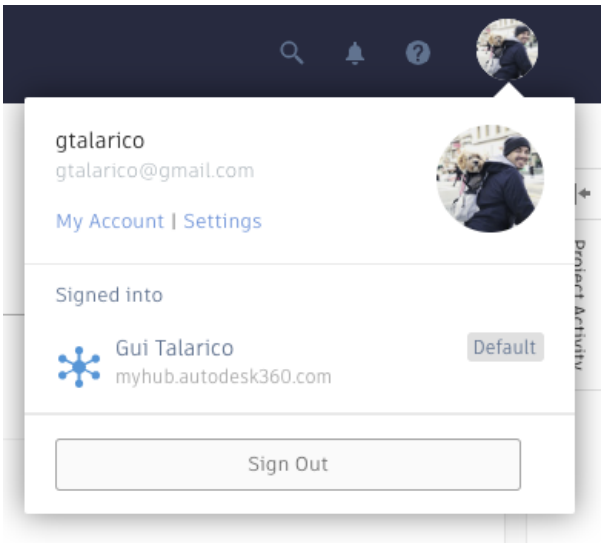
- GET projects/:project_id/folders/:folder_id/contents

  Get the contents of a folder

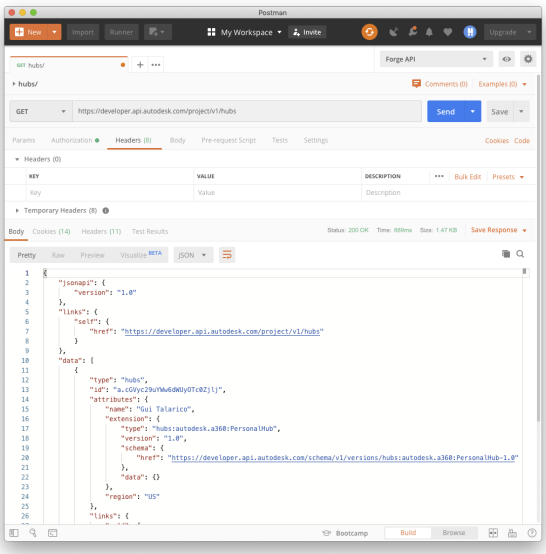
- GET projects/:project_id/items/:item_id

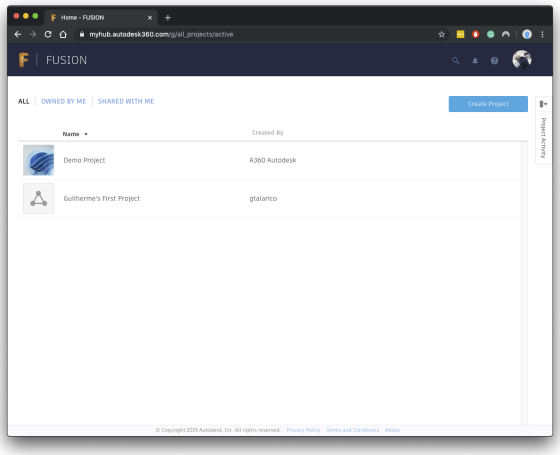
  Get an item in a folder

Try going through each one of these on your own. It's helpful to save the target id of each resource in a variable so you can use in the subsequent requests - ie. after fetching hubs, save one of the hub ids in a variable `HUB_ID` so you can reference it when you fetch `hubs/:hub_id/projects`

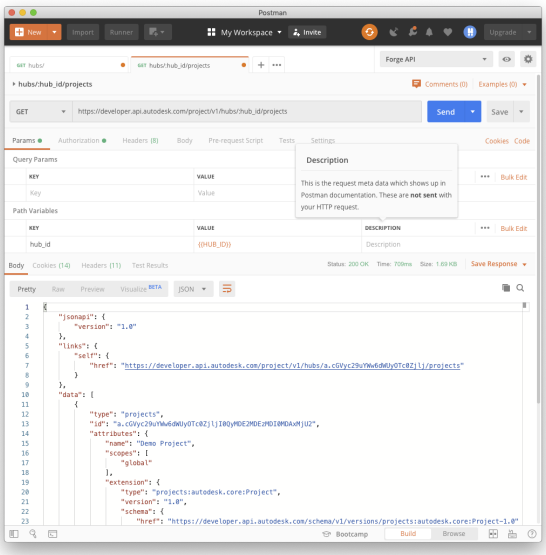


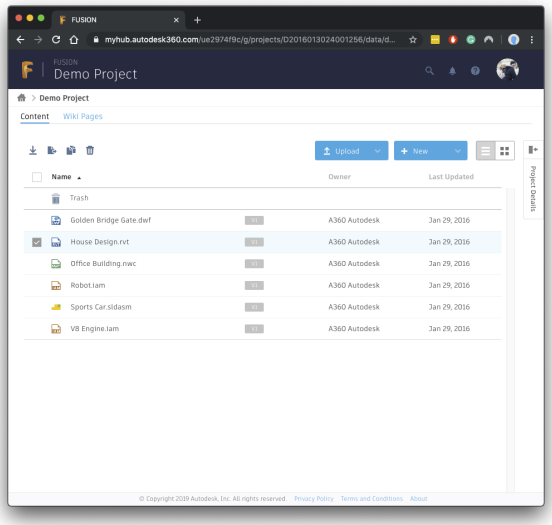
List of User Hubs



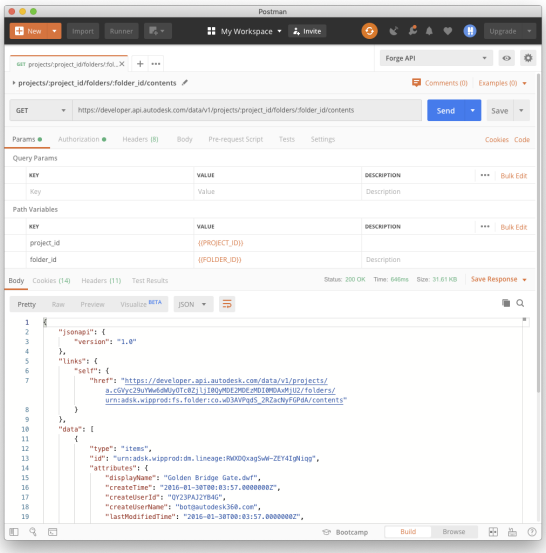


List of Projects in a Hub





List of Items in a folder



Other Tutorials

- How to Upload a File
- Publish a Cloudshared Revit Model

Using the API within an Application

We have been able to use a variety of endpoints of the Airtable and Forge API, so what now?

Tools like cUrl and Postman are great to test and explore Apis, but be able to use make real those APIS we need to use them within the context of an application.

Building an application is outside the scope of this session, but I will leave you some resources and references. If you already know how to write code, you should be ready to start using Rest APIs like Airtable and Forge.

If your are still learning how to code, writing an application that interacts with APIs is a great project to learn.

- https://forge.autodesk.com/code-samples
- https://github.com/gtalarico/au-2019-web-apis
- https://github.com/cyrillef/forge.data.management-js
- https://forgedatamanagement.herokuapp.com/