Intro to Web Apis

Gui Talarico

AUTODESK UNIVERSITY

Intro to Web Apis

Autodesk University 2019

Gui Talarico
Software Engineer
Samara

@gtalarico

Silman

LAGARSOFT
INFORMATION TECHNOLOGY

Timon Hazell
Head of Computational Design
Silman

Pablo Gancharov
Founder
Lagarsoft

github.com/gtalarico/au-2019-web-apis

gitter.im/gtalarico/au-2019-web-apis

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.

- 💡 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

* Basic programming experience in any language is helpful but not required

Requirements

- ❤️ Postman
- ❤️ Cmder
- ❤️ Airtable Account
- ❤️ Autodesk Account

Tooling

cUrl
Command Line
HTTP Client

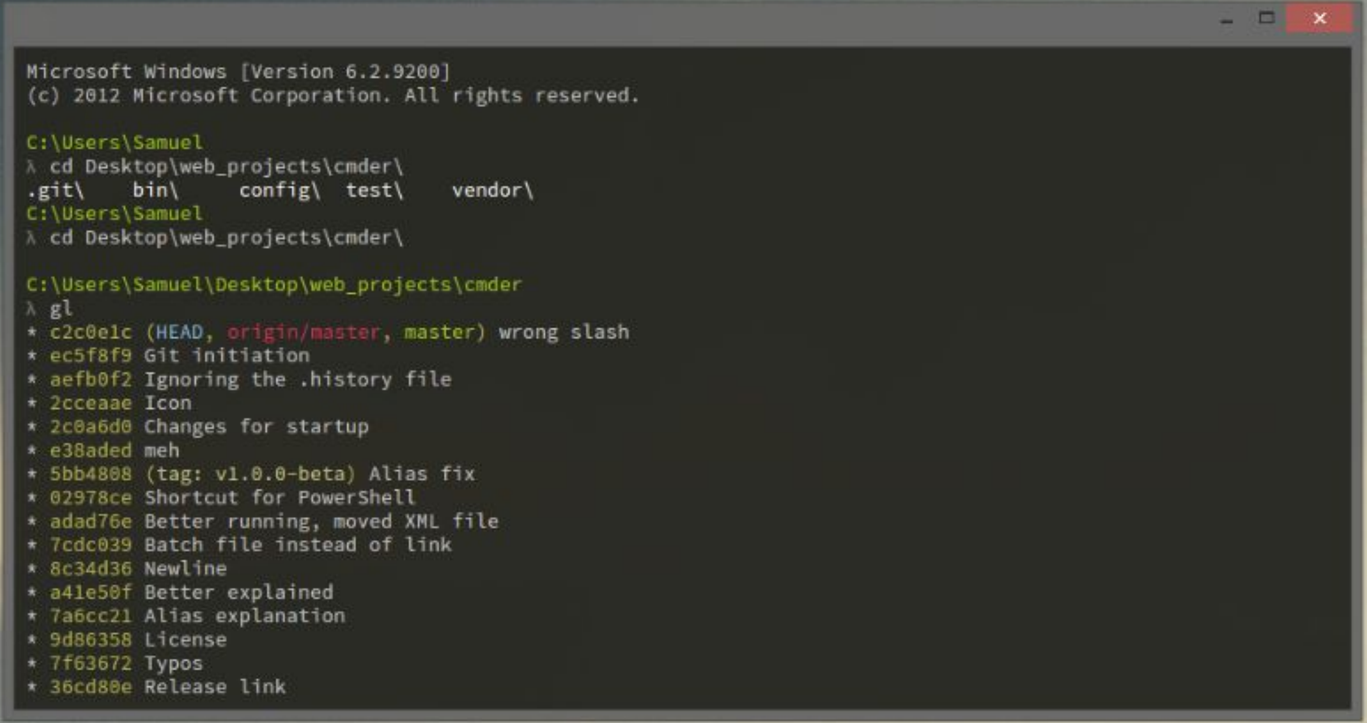
Postman
API Testing and
Development Environment

Chrome DevTools
Tools for Web Developers
built into the Chrome
Browser

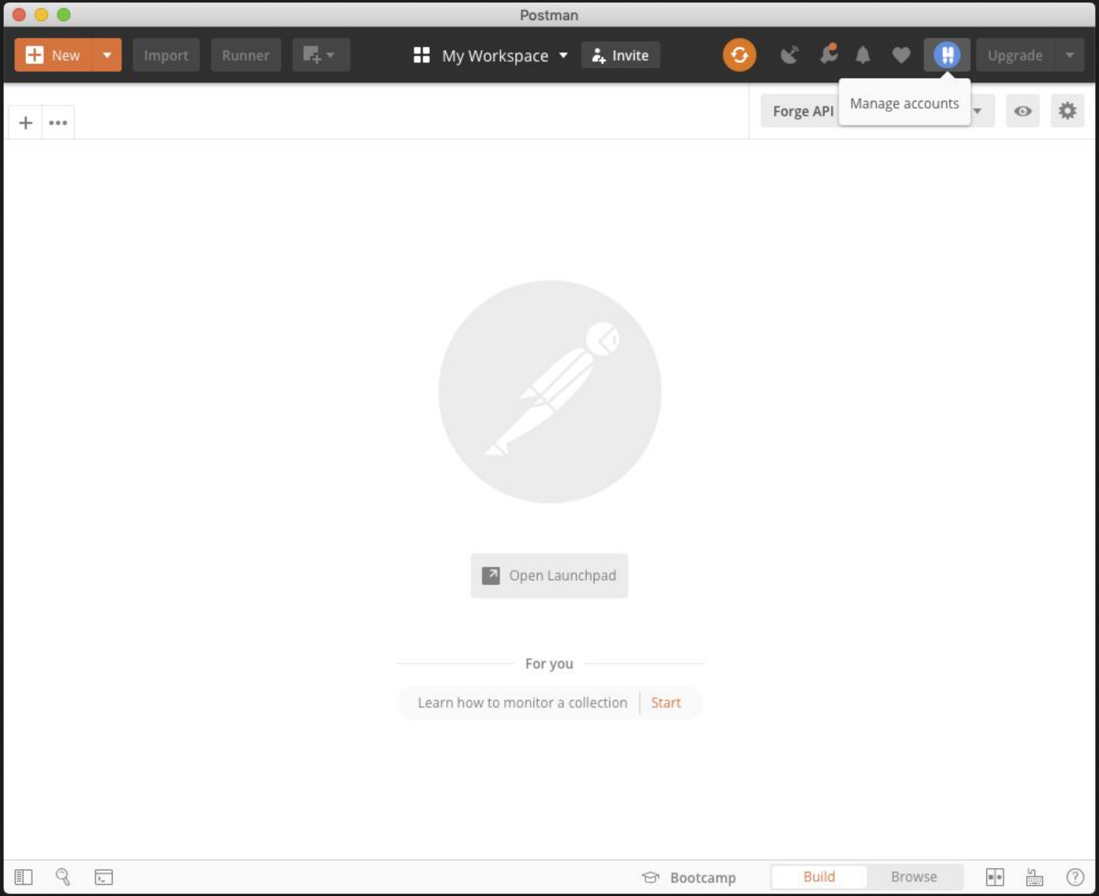
Python + Requests
Easy to use Interpreted
language

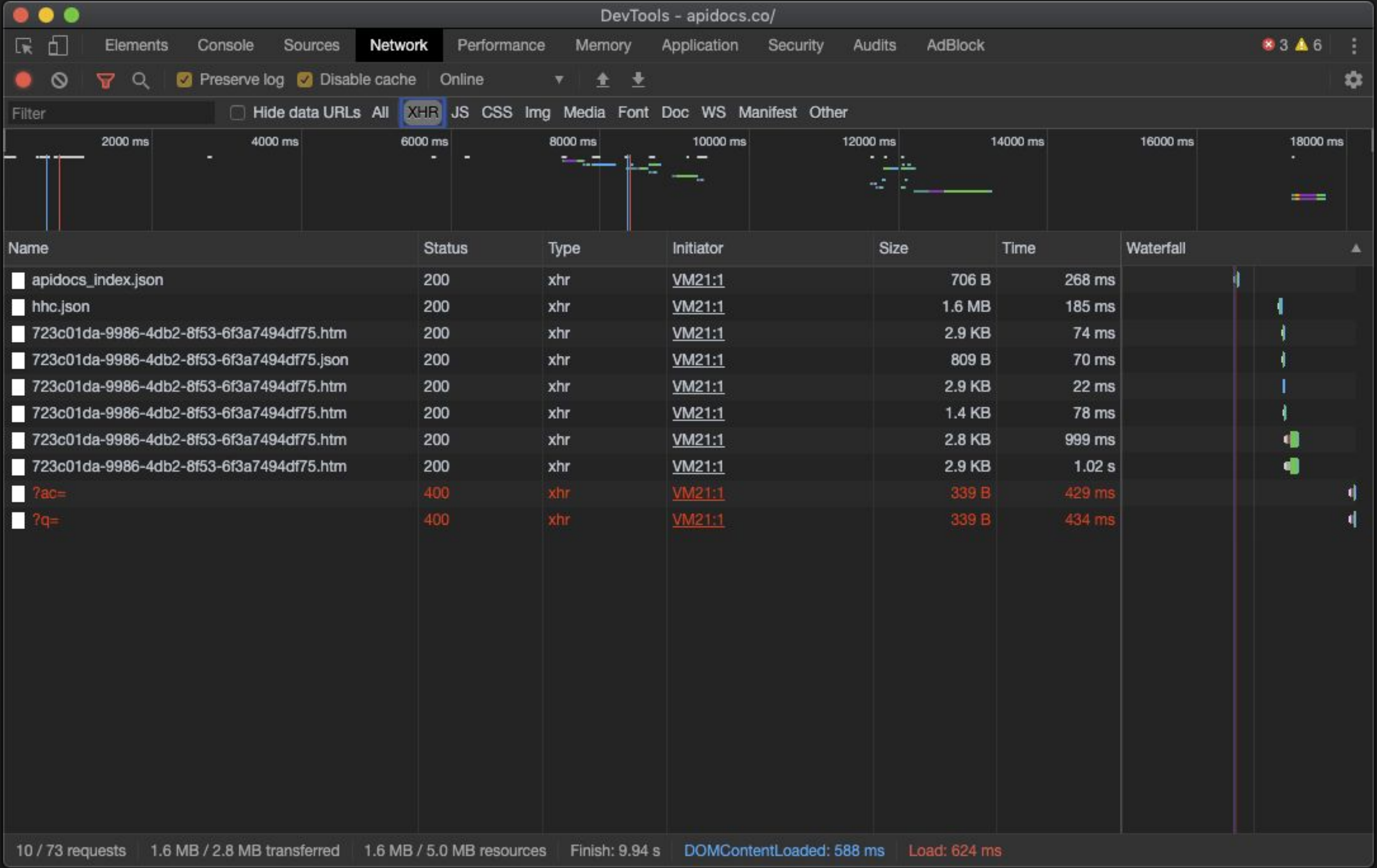
λcmder

Portable console emulator for Windows



```
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\Samuel
λ cd Desktop\web_projects\cmder\
.git\    bin\     config\  test\    vendor\
C:\Users\Samuel
λ cd Desktop\web_projects\cmder\

C:\Users\Samuel\Desktop\web_projects\cmder
λ gl
* c2c0e1c (HEAD, origin/master, master) wrong slash
* ec5f8f9 Git initiation
* aefb0f2 Ignoring the .history file
* 2cceaae Icon
* 2c0a6d0 Changes for startup
* e38aded meh
* 5bb4808 (tag: v1.0.0-beta) Alias fix
* 02978ce Shortcut for PowerShell
* adad76e Better running, moved XML file
* 7cdc039 Batch file instead of link
* 8c34d36 Newline
* a41e50f Better explained
* 7a6cc21 Alias explanation
* 9d86358 License
* 7f63672 Typos
* 36cd80e Release link
```


Postman
New
Import
Runner
My Workspace
Invite
Upgrade
Manage accounts
Forge API
Open Launchpad
For you
Learn how to monitor a collection
Start
Bootcamp
Build
Browse



DevTools - apidocs.co/

Name	Status	Type	Initiator	Size	Time	Waterfall
apidocs_index.json	200	xhr	VM21:1	706 B	268 ms	
hhc.json	200	xhr	VM21:1	1.6 MB	185 ms	
723c01da-9986-4db2-8f53-6f3a7494df75.htm	200	xhr	VM21:1	2.9 KB	74 ms	
723c01da-9986-4db2-8f53-6f3a7494df75.json	200	xhr	VM21:1	809 B	70 ms	
723c01da-9986-4db2-8f53-6f3a7494df75.htm	200	xhr	VM21:1	2.9 KB	22 ms	
723c01da-9986-4db2-8f53-6f3a7494df75.htm	200	xhr	VM21:1	1.4 KB	78 ms	
723c01da-9986-4db2-8f53-6f3a7494df75.htm	200	xhr	VM21:1	2.8 KB	999 ms	
723c01da-9986-4db2-8f53-6f3a7494df75.htm	200	xhr	VM21:1	2.9 KB	1.02 s	
?ac=	400	xhr	VM21:1	339 B	429 ms	
?q=	400	xhr	VM21:1	339 B	434 ms	

10 / 73 requests | 1.6 MB / 2.8 MB transferred | 1.6 MB / 5.0 MB resources | Finish: 9.94 s | DOMContentLoaded: 588 ms | Load: 624 ms

Terminology

Client

An application that access resources and services made available by a server

aka a Consumer

Server

An application that performs functionality and/or serves resources consumed by a client.

API

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

Web API

An API that provides functionality by means of requests over the internet, most commonly through HTTP requests.

The expected request and response formats define a contract between the client and server

Web Resource

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

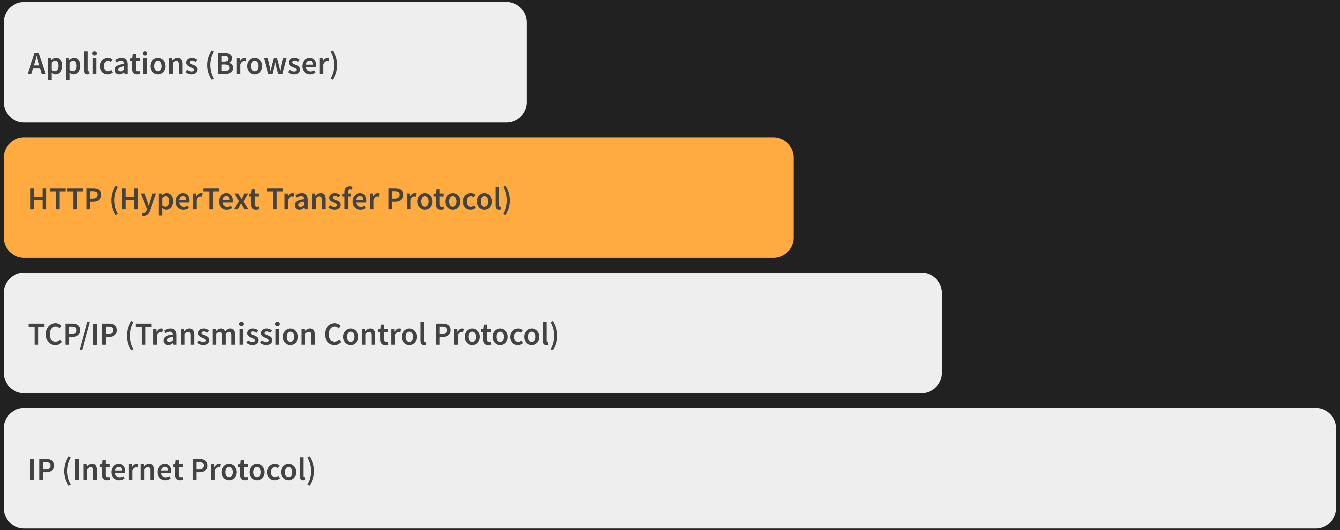
Common web resources are html pages,
media (images, videos), files (zip, exe), JSON and XML objects, etc.

HTTP Protocol

HTTP is a protocol which allows the fetching of resources such as HTML documents.

A client-server protocol and the foundation of data exchange on the Web.

HTTP Protocol


Applications (Browser)
HTTP (HyperText Transfer Protocol)
TCP/IP (Transmission Control Protocol)
IP (Internet Protocol)

HTTP Protocol

HTTP or Hypertext Transfer Protocol is 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)

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

HTTP 1.0 x HTTP 2.0

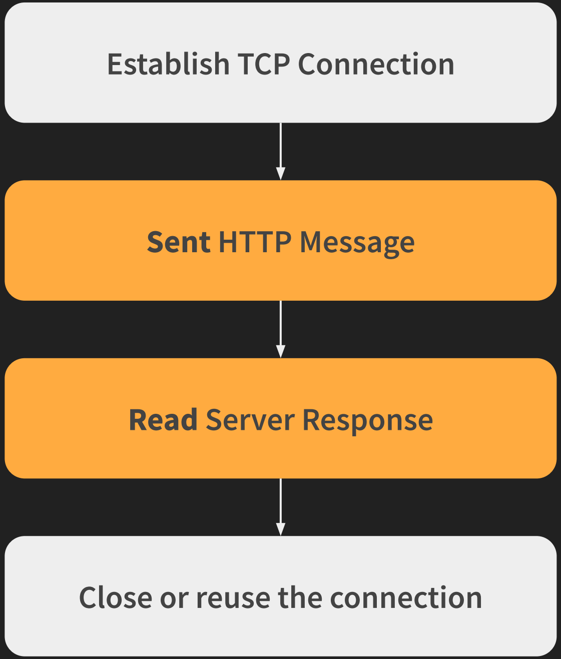
HTTP 2.0 is the most recent protocol and was published in 2014.

It’s 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

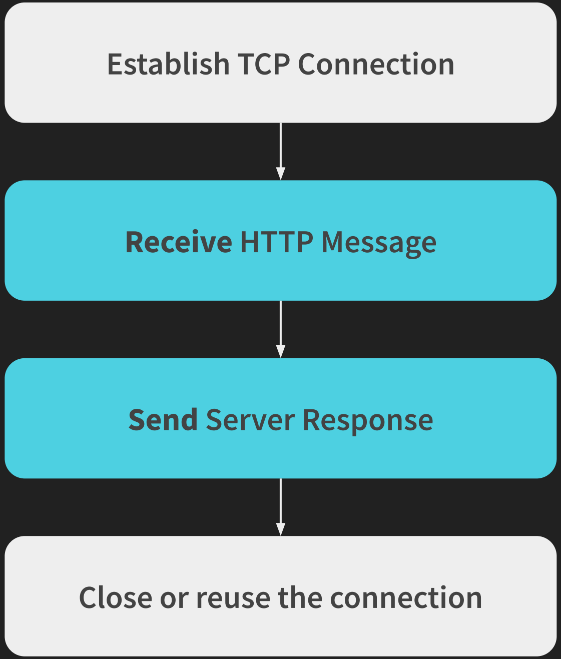
https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

HTTP FLOW



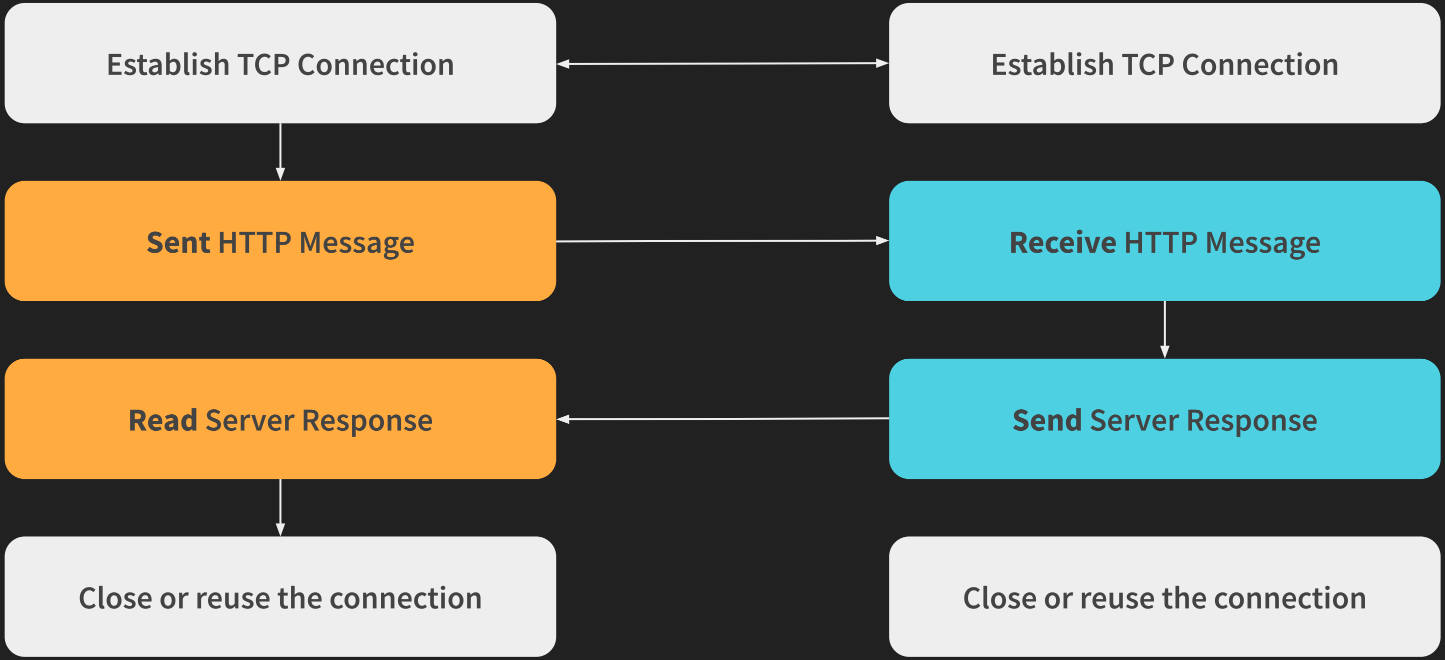
Client Side

HTTP FLOW



Server Side

HTTP FLOW


Establish TCP Connection
Establish TCP Connection
Sent HTTP Message
Receive HTTP Message
Read Server Response
Send Server Response
Close or reuse the connection
Close or reuse the connection

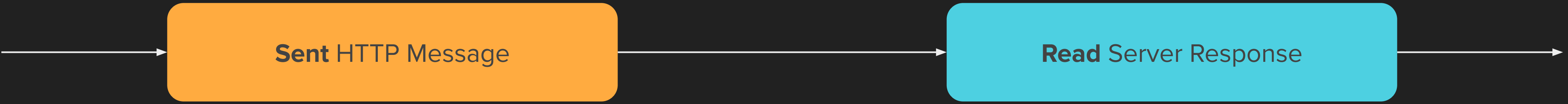
HTTP Message Format

Client Request Message

```
GET /users/ HTTP/1.1
Host: www.server.com
Accept: */*
```

Server Response Message

```
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>Page Content</body></html>
```



Client Request Message

```
GET /users/ HTTP/1.1
Host: www.server.com
Accept: */*
```

```
{METHOD} /{RESOURCE} HTTP/{VERSION}
Host: {SERVER ADDRESS}
Accept: {Format}
{BLANK LINE}
{PAYLOAD}
```

Server Response Message

```
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>Page Content</body></html>
```

```
{PROTOCOL} {RESPONSE STATUS}
...
Content-Type: {format: html, json, media}
{BLANK LINE}
{RESPONSE BODY}
```

GET

Requests a resource.
Requests using GET should only retrieve data.
Requests should be *Idempotent

* I·dem·po·tent
the property of certain operations [...] whereby they can be applied multiple times without changing the result beyond the initial application
Wikipedia

```
GET /{resource} HTTP/1.1
Host: www.server.com
```

HTTP Methods

Client Request Message

```
GET / HTTP/1.1
Host: www.google.com
```

Server Response Message

```
HTTP/1.1 200 OK
Content-Type: text/html

<html><body>Page Content</body></html>
```

Client Request Message

```
GET /users/?name=Gui Talarico HTTP/1.1
Host: www.server.com
Accept: application/json
```

Server Response Message

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
      “user_id”: 1,
      “name”: “Gui Talarico”
  }
]
```

* Note Url Parameter

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) and are not idempotent.

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

```
POST /users/ HTTP/1.1
Host: www.server.com
Content-type: {format}

{data}
```

HTTP Methods

Client Request Message

```
POST /users/ HTTP/1.1
Host: www.server.com
Content-Type: application/x-www-form-urlencoded

name=Gui+Talarico
age=16
```

Server Response Message

```
HTTP/1.1 201 OK
Content-Type: application/json

{
    “errors”: [],
    “user_id”: 1
}
```

Client Request Message

```
POST /users/ HTTP/1.1
Host: www.server.com
Content-Type: application/json

{
    “name”: “Gui Talarico”,
    “age”: 16
}
```

Server Response Message

```
HTTP/1.1 201 OK
Content-Type: application/json

{
    “errors”: [],
    “user_id”: 1
}
```

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.”

```
PUT /users/123 HTTP/1.1
Host: www.server.com
Content-Type: application/json

{“name”: “Gui Talarico”, “admin”: true}
```

HTTP Methods

DELETE

“The HTTP DELETE request method deletes the specified resource.”

```
DELETE /users/123 HTTP/1.1
Host: www.server.com
```

HTTP Methods

Status Codes

Status Codes

HTTP response status codes indicate whether a specific HTTP request has been successfully completed.

Responses are grouped in five classes:

100’s: Informational responses
200’s: Successful responses
300’s: Redirects
400’s: Client errors
500’s: Servers errors

Status codes are defined by section 10 of RFC 2616

Status Codes

200 OK
The request has succeeded.

201 CREATED
Your request resulted in the creation of new resource.

301 MOVED
Target resource has moved

400 BAD REQUEST
Server could not understand your request (malformed, invalid)

403 FORBIDDEN
Request understood but will not authorize, sending same request will result in the same response

404 NOT FOUND
The target request could not be found

418 IM A TEA POT
Easter Egg that was never removed from the Specs

500 INTERNAL SERVER ERROR
Unexpected server error

Authentication

Basic

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

```
# python
>>> credentials = base64.b64encode(b‘user:pwd’)
>>> print(b64credentials)
b'dXNlcm5hbWU6cGFzc3dvcmQ='
```

```
GET /users/123 HTTP/1.1
Host: www.server.com
Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=
```

Authentication

Bearer / Oauth

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).

```
GET /users/123 HTTP/1.1
Host: www.server.com
Authorization: Bearer {token}
```

Oauth 2-legged

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).

Authentication

Client

```
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
```

Server

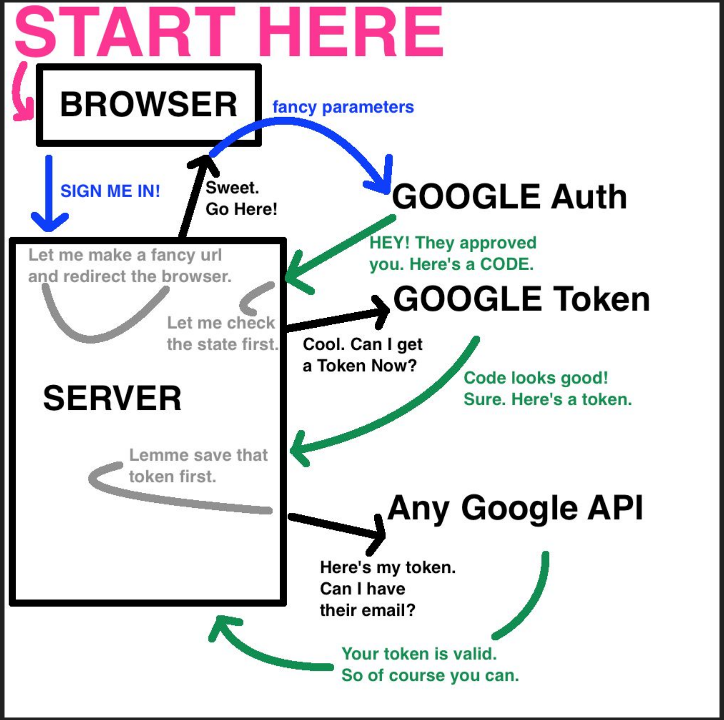
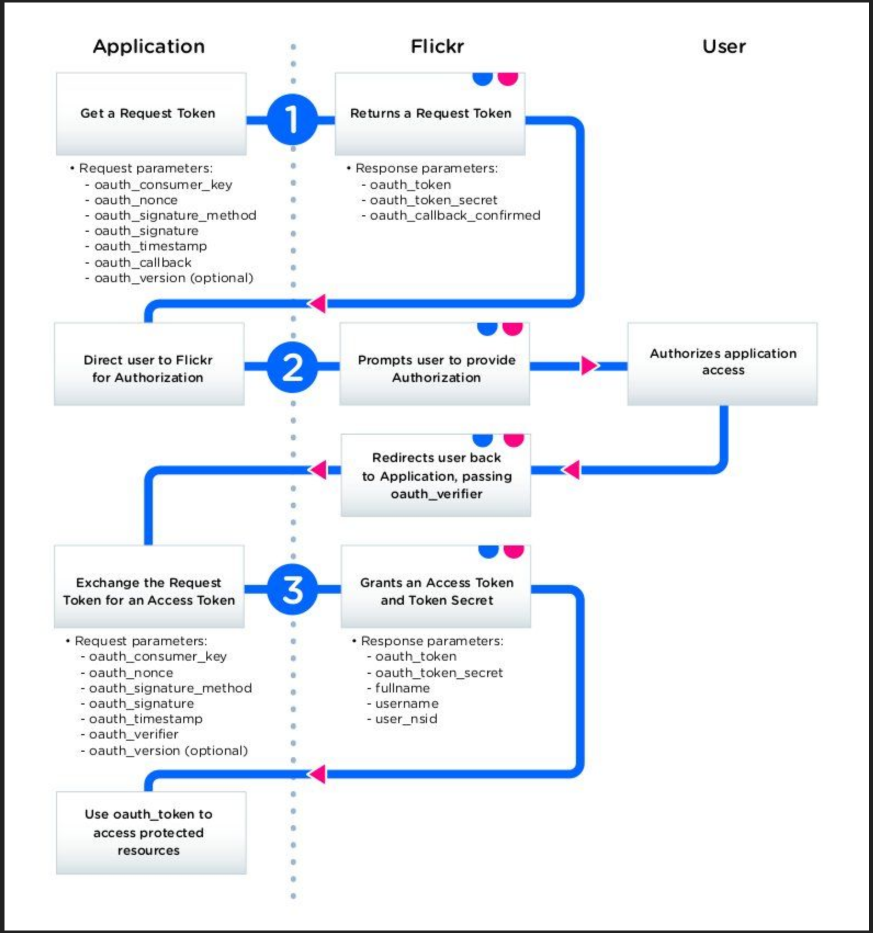
```
HTTP/1.1 200 OK
Content-Type: application/json
[ other headers ]

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

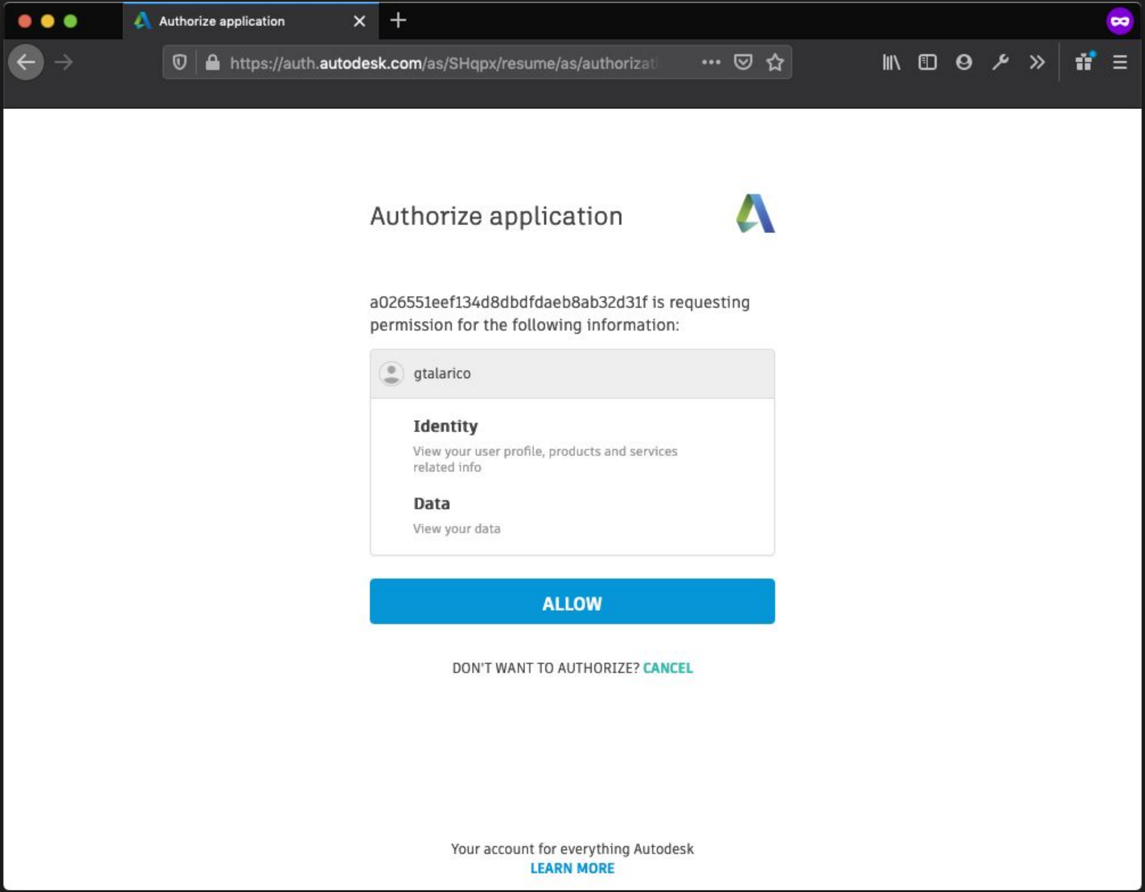
```
POST /secure-resource HTTP/1.1
Host: developer.api.autodesk.com
Content-Type: application/x-www-form-urlencoded

access_token: eyJhbGc … 8B2nSjrq_ys
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{ secure data }
```

Oauth 3-legged

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


START HERE
BROWSER
fancy parameters
SIGN ME IN!
Sweet.
Go Here!
GOOGLE Auth
Let me make a fancy url and redirect the browser.
HEY! They approved you. Here's a CODE.
Let me check the state first.
GOOGLE Token
Cool. Can I get a Token Now?
Code looks good! Sure. Here's a token.
SERVER
Lemme save that token first.
Any Google API
Here's my token. Can I have their email?
Your token is valid. So of course you can.

Application
Flickr
User
Get a Request Token
1
Returns a Request Token
• Request parameters:
- oauth_consumer_key
- oauth_nonce
- oauth_signature_method
- oauth_signature
- oauth_timestamp
- oauth_callback
- oauth_version (optional)
• Response parameters:
- oauth_token
- oauth_token_secret
- oauth_callback_confirmed
Direct user to Flickr for Authorization
2
Prompts user to provide Authorization
Authorizes application access
Redirects user back to Application, passing oauth_verifier
Exchange the Request Token for an Access Token
3
Grants an Access Token and Token Secret
• Request parameters:
- oauth_consumer_key
- oauth_nonce
- oauth_signature_method
- oauth_signature
- oauth_timestamp
- oauth_verifier
- oauth_version (optional)
• Response parameters:
- oauth_token
- oauth_token_secret
- fullname
- username
- user_nsid
Use oauth_token to access protected resources


Authorize application
https://auth.autodesk.com/as/SHqpx/resume/as/authorizati

Authorize application

a026551eef134d8dbdfdaeb8ab32d31f is requesting permission for the following information:

gtalarico

Identity

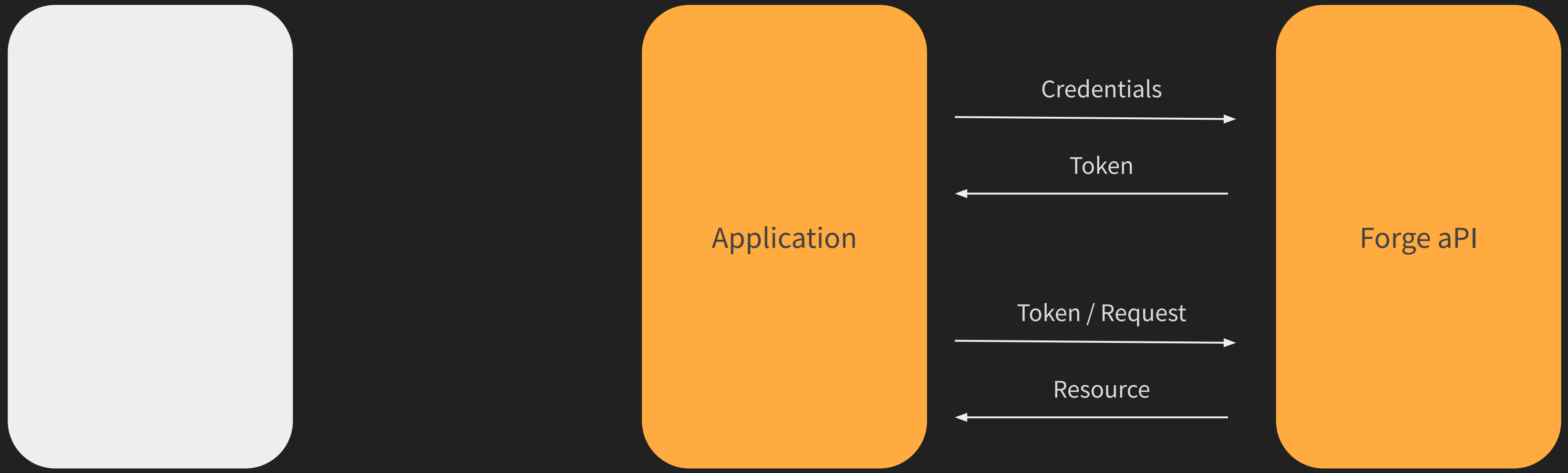
View your user profile, products and services related info

Data

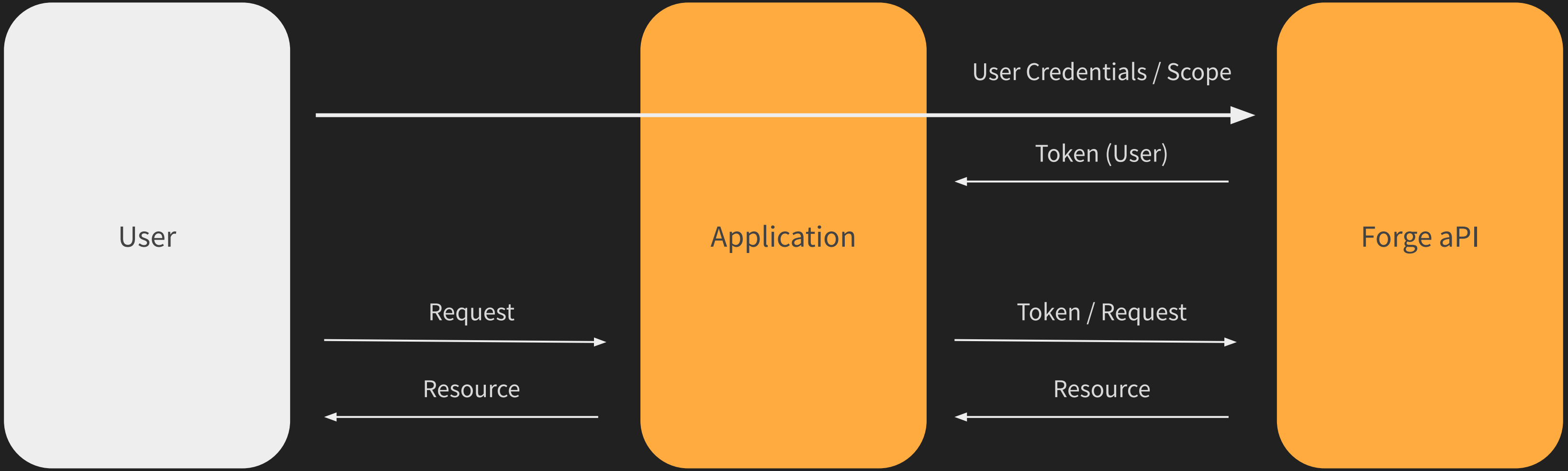
View your data

ALLOW
DON'T WANT TO AUTHORIZE? CANCEL
Your account for everything Autodesk
LEARN MORE

2-Legged


Credentials
Token
Application
Forge aPI
Token / Request
Resource

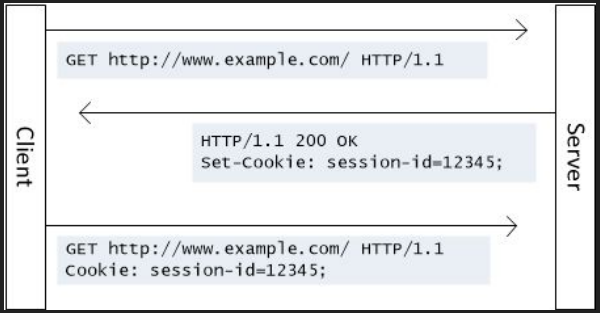
3-Legged


User Credentials / Scope
Token (User)
User
Application
Forge aPI
Request
Token / Request
Resource
Resource

Cookies

HTTP is Stateless

Cookies allow the server to maintain state by telling the client to store data, and re-send it on subsequent requests.



```
GET http://www.example.com/ HTTP/1.1
```

```
HTTP/1.1 200 OK
Set-Cookie: session-id=12345;
```

```
GET http://www.example.com/ HTTP/1.1
Cookie: session-id=12345;
```

Cookies

Client 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=appie-pie
Set-Cookie: session_id=zxczxcasd; Expires=Wed, 21 Oct 2015 07:28:00 GMT;
```

Client Request

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

REST

Representational State Transfer

REST is an architectural style that defines a set of constraints to be used for creating web services.

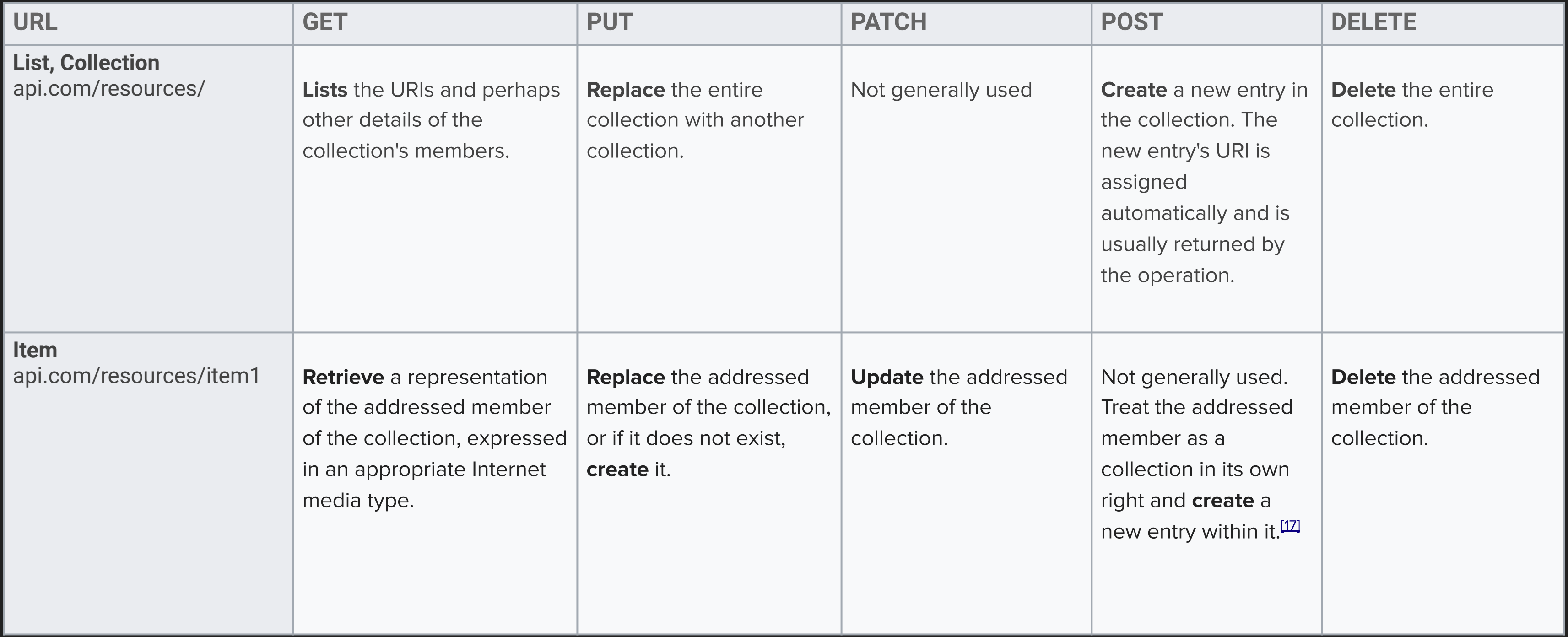
HTTP Object Model

```
GET     /resources/                   < Get Collection
GET     /resources/item1              < Get Item
DELETE /resources/item1               < Delete Item
POST    /resources/                   < Create item
GET    /resources/item1/details       < Get Item Details
```

Not RESTful

```
GET     /getitems/new
POST    /createnew/1
```

REST



URL	GET	PUT	PATCH	POST	DELETE
List, Collection api.com/resources/	**Lists** the URIs and perhaps other details of the collection's members.	**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.
Item 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.

Hands On

Airtable API

Airtable API

1. Go to airtable.com/api
2. Login or Create an Account

3. Read the Documentation
4. Make GET and PATCH Requests on Postman
5. Analyze Postman Code

6. Make the same Request using cUrl
7. Make the same Request using Python + Requests

Forge

1. Create Forge App
2. Authenticate as an Application
3. Verify Authentication

4. Authenticate as a User
5. Verify Authentication
6. Fetch Resources on behalf of User
    a. Hubs
    b. Projects
    c. Folders

7. Show how requests are user on an Application

Forge API

What’s Next

1. Consume API through scripts (Python, IronPython, JS, C#)
2. Create Applications

Next

Forge Applications

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/

thank you

@gtalarico
twitter

`github.com/gtalarico/au-2019-web-apis`