

Bio



Mike Stackhouse is the Chief Innovation Office at atorus. He previously spent 6 years at Covance in statistical programming. He started with SAS programming and found his way into working however he could with languages like R and Python.

Mike is a co-lead of the phuse working group project Open Source Technologies in Clinical Research. Their focus is to provide the industry with use cases, best practices, and implementation and validation guidance in order to increase the adoption of open source technology in our regulatory landscape.

In May 2020, Mike expects to graduate from the master's of information and data science (MIDS) program at UC Berkeley School of Information, where he has worked on projects involving computer vision, natural language processing, cluster computing, and deep learning. His special interests include automation, machine learning, big data technology, and mentoring rising programmers.



Opening Doors for Automation with Python and REST:

A SharePoint Example

Mike Stackhouse

TT16

Agenda

- Motivation
- What is a REST API?
 - Anatomy of a web request
- Web Requests in Python
- Authentication
- SharePoint and REST
 - Download a file
 - Download a list of files
 - Upload a file

Motivation

Motivation



- Push to automate and trim process fat
- Common roadblock is connecting systems
 - We have data there, but the systems don't talk
- People outside the SAS group benefit from automation too
- Languages like Python can be made available to anyone

What is a REST API?

What is a REST API?

- API = **A**pplication **P**rogramming **I**nterface
 - Abstract away complicated parts of a process so programs/systems can “talk”
- REST = **R**Epresentational **S**tate **T**ransfer
 - Set of rules that developers follow when creating an API

What is a REST API?

Anatomy of a Web Request

- REST APIs work through URLs
- URLs return a piece of data (a *resource*)
- Each URL is called a *request* – data you send back is called a *response*
- Requests are made of 4 parts:
 1. The **endpoint**
 2. The **method**
 3. The **headers**
 4. The **data**

What is a REST API?

The Endpoint

- The URL from which you request data
- Can be broken into parts
 - Root endpoint
 - e.g. ClinicalTrials.gov API endpoint = <https://www.clinicaltrials.gov/ct2/>
 - The path – similar in structure to a directory.
 - e.g. <https://www.clinicaltrials.gov/ct2/some/path/>
 - Need to consult API documentation to understand a paths meaning
 - Query parameters:
 - `www.somesite.com/api/path/?param1=value1¶m2=value2`

Note: Query parameter are not technically part of the REST architecture, but APIs commonly use them

What is a REST API?

The Endpoint - Example



In this query, we are:

- Requesting clinical trial results from the API
- Downloading the data automatically from the request
- Filtering for only trials where the condition is 'cancer'
- Downloading only 10 records
- Downloading in CSV format

https://clinicaltrials.gov/ct2/results/download_fields?cond=cancer&down_count=10&down_fmt=csv

What is a REST API?

The Method

- 5 different request methods – GET, POST, PUT, PATCH, and DELETE
- We will focus on only two:

GET	Get a resource from a server. The server looks for the data you requested and sends it back to you. The server performs a READ action.
POST	Create a new resource on a server. The server creates a new entry in the database and tells you whether the operation was successful. The server performs a WRITE action.

What is a REST API?

The Headers

- Provide information about the client and the server
- NOT in the URL itself
 - Sent along with the request
- Import for several reasons, but two pertinent:
 - Tell the server how you want to receive information
 - Authentication
- Property-value pairs
`Accept: application/json`

What is a REST API?

The Data

- Data you want to be sent to the server
- Not necessary in a GET request
- Required for a POST request

Web Requests with Python

Web Requests with Python



- Any programming language capable of web requests can use a REST API
- Python makes web requests easy

```
>>> import requests
>>> r = requests.get('https://clinicaltrials.gov/ct2/results/download_fields?
cond=cancer&down_count=1000&down_fmt=csv')
```

Web Requests with Python

Status Codes

- Code meanings
 - Successful responses 200-299
 - Client errors (your mistake) 400-499
 - Server errors (their mistake) 500-599

400 Bad Request	Something about the syntax of your request was incorrect – you likely have a mistake in your URL
401 Unauthorized	The server does not know who you are, but needs to for you to receive your desired request
403 Forbidden	The server knows who you are, but you do not have the proper permissions to receive the request

```
>>> print(r.status_code)
```

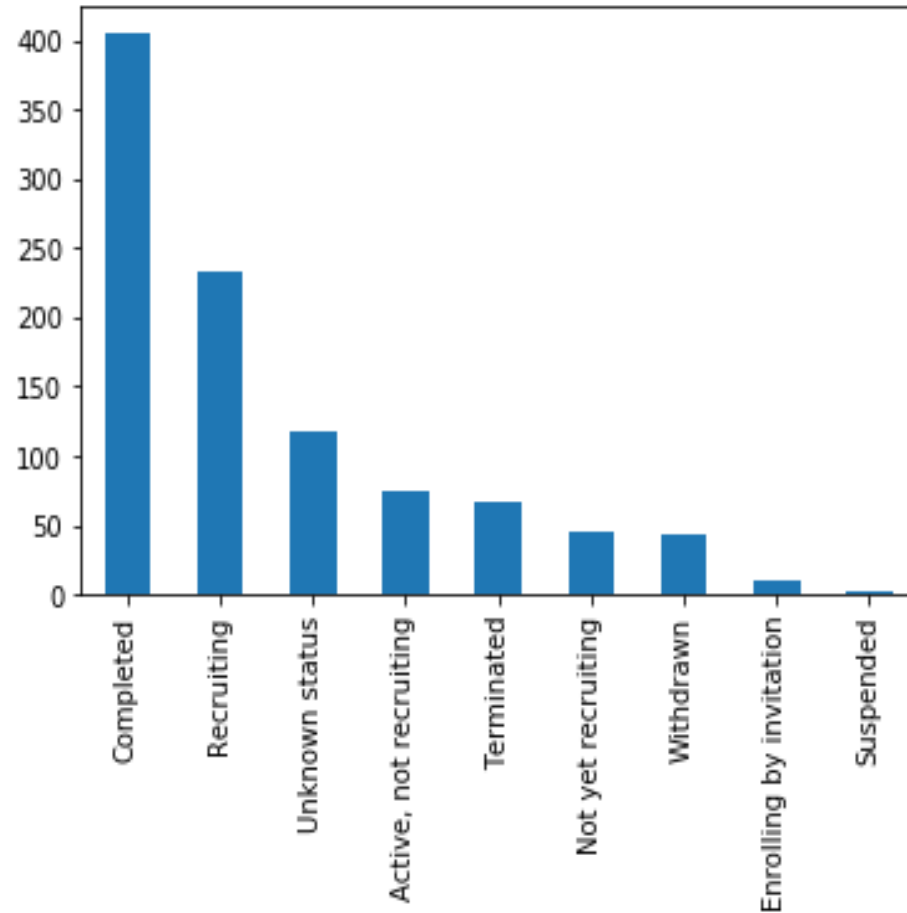
```
200
```

Web Requests with Python

Accessing Data

- Stored in two attributes
 - text – data as a string
 - Content – data in binary form

```
>>> import io  
>>> import pandas as pd  
  
>>> df = pd.read_csv(io.BytesIO(r.content))  
>>> df['Status'].value_counts().plot(kind='bar')
```



Authentication

Authentication

A 10,000 mile view



- Provides a barrier between your request and the server to make sure you have permission to view the resource
- Simple case – HTTP Basic Authentication
 - User ID: USER
 - Password: PASSWORD
 - HTTP Request Header
 - Authorization: Basic VVNFUjpQQVNTV09SRA==

```
>>> import requests
>>> from requests.auth import HTTPBasicAuth
>>> requests.get('www.someurl.com', auth=HTTPBasicAuth('USER', 'PASSWORD'))
```

^^^ *Don't use this...*

Authentication

A 10,000 mile view



Here's why

```
>>> import base64  
  
>>> base64.b64decode('VVNFUjpQQVNNTV09SRA==')  
  
b'USER:PASSWORD'
```

Not very secure...

Authentication

A 10,000 mile view



- Something better – NTLM authentication

```
>>> import requests
```

```
>>> from requests_ntlm import HttpNtlmAuth # Not in base library
```

```
>>> requests.get('www.someurl.com', auth=HttpNtlmAuth('USER', 'PASSWORD'))
```

- For SharePoint, another option: SAML Token Service for Office 365
 - Instead of re-authenticating you at every request, header request is updated with cookies that contain authorization tokens that last the life of their session
 - Once session completes, cookies are no longer valid and user has to re-authenticate

SharePoint and REST

SharePoint and REST

Overview



- Able to interface with most of the application at a granular level
- Most of this out of scope, but we will explore three basic functions that assist automation
 - Listing the files in a SharePoint folder
 - Downloading files
 - Uploading files
- API is more complicated than what we've covered so far

SharePoint and REST

spLite



- Python modules available to interface with the full Office 365 REST API
 - [Office365-REST-Python-Client](#)
 - Full featured API
 - MIT License
- I've released [spLite](#)
 - Lightweight module that covers just the previously mentioned functionality
 - Covers the authentication methods previously discussed
 - Written to let the user easily accomplish basic automation tasks
 - MIT License

```
>>> from spLite import sharepoint
>>> s = sharepoint.SpSession(<site_url>,<username>,<password>)
# Choose your authentication type
>>> s = sharepoint.SpSession(<site_url>,<username>,<password>, context_type='ntlm')
>>> s = sharepoint.SpSession(<site_url>,<username>,<password>, context_type='basic')
```

SharePoint and REST

Download a File



GET

```
https://{site_url}/_api/web/GetFolderByServerRelativeUrl('{folder_name}')/Files('{file_name}')/$value
```

site_url	The root-endpoint of your SharePoint site. This should be the home page of your SharePoint site (so it may not necessarily end in .com)
folder_name	The subfolder of the SharePoint site that your files resides in. This can occasionally be tricky – because the displayed folder name may be different than the link necessary. For example, ‘Documents’ may actually be ‘Shared Documents’. Also – note that the text must be valid in a URL For example, spaces aren’t valid, so ‘Shared Documents’ should be ‘Shared%20Documents’. You can quote from the library <code>urllib.parse</code> in Python to ensure your string is formatted correctly.
file_name	The file name of the file that you want to download. Again, make sure that the string is valid in the URL as described in folder name.

SharePoint and REST

Download a File



- Read a file into memory

```
>>> s.get_file(<folder>,<file>)
```

- Read a file and write to disk

```
>>> s.get_file(<folder>,<file>, output_location=<folder path>)
```

SharePoint and REST

Download a List of Files



GET
`https://{site_url}/_api/web/GetFolderByServerRelativeUrl('{folder_name}')/Files`

- All files in a folder

```
>>> s.list_and_get_files(<folder>, output_location=<folder path>)
```

- Just a subset of files by name:

```
>>> s.list_and_get_files(<folder>, output_location=<folder path>, files=['a.txt', 'b.txt'])
```

- Subsets by extensions

```
>>> s.list_and_get_files(<folder>, output_location=<folder path>, extensions='.xlsx')
```

```
>>> s.list_and_get_files(<folder>, output_location=<folder path>, extensions=['.xlsx', '.xls'])
```

SharePoint and REST

Upload a file



POST

```
https://{site_url}/_api/web/GetFolderByServerRelativeUrl('{folder_name}')/Files/add(url='a.txt',overwrite=true)
```

- Upload a file

```
>>> s.upload_file(<folder>, <file>)
```

- Avoid overwriting

```
>>> s.upload_file(<folder>, <file>, overwrite=False)
```

- Three ways a file can be provided

- File path (str object)
- A bytes object
- A file-like object that has a read method

SharePoint and REST

Upload a file



- Upload a bytes object

```
>>> import io
>>> my_string = 'Here is some text'
>>> data = bytes(my_string, encoding='utf8')
>>> s.upload_file('some/folder', 'some_file.txt', data=data, overwrite=False)
```

- Upload a file-like object

```
>>> import pandas as pd
>>> data = df.to_csv() # This assume df existed and had data
>>> file = io.BytesIO(bytes(data, encoding='utf8'))
>>> s.upload_file('some/folder', 'my_data.csv', data=file, overwrite=False)
```

Conclusion

Conclusion



- REST APIs can open connection otherwise thought closed
- Use cases
 - Automate report generation to stakeholders without access to our servers
 - Pull data from users who don't have access to our servers
 - Put a tool in the hands non-technical users, like a project manager
- spLite is freely available for you to use

Questions?

Mike Stackhouse

Atorus Research

Mike.Stackhouse@atorusresearch.com

<https://www.linkedin.com/in/michael-s-stackhouse/>

References

1. Stackhouse, M. (2020, January 21). mstackhouse/spLite. Retrieved from <https://github.com/mstackhouse/spLite>
2. Zell, & Liew. (2018, January 17). Understanding And Using REST APIs. Retrieved from <https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/>
3. Downloading Content for Analysis. (n.d.). Retrieved from <https://clinicaltrials.gov/ct2/resources/download>
4. (n.d.). Retrieved from <https://www.reddit.com/dev/api/>
5. HTTP headers. (n.d.). Retrieved from <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>
6. HTTP response status codes. (n.d.). Retrieved from <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>
7. Spdevdocs. (n.d.). Get to know the SharePoint REST service. Retrieved from <https://docs.microsoft.com/en-us/sharepoint/dev/sp-add-ins/get-to-know-the-sharepoint-rest-service?tabs=csom>
8. Gremyachev, V. (2020, January 13). vgreem/Office365-REST-Python-Client. Retrieved from <https://github.com/vgreem/Office365-REST-Python-Client>
9. Spdevdocs. (n.d.). Work with __REQUESTDIGEST. Retrieved from <https://docs.microsoft.com/en-us/sharepoint/dev/spfx/web-parts/basics/working-with-requestdigest>