Configuring programs and using APIs

Data Sciences Institute
University of Toronto

Simeon Wong

What is an API?

- API stands for Application Programming Interface
- Allows software programs to communicate with each other
- Provides structured way to communicate between applications and devices (expose data and functionality)
- Allows other developers to access and integrate with an application without needing to understand complex implementation details

Programs using programs

- You've already used APIs!
- Python APIs: matplotlib, numpy, pandas
- Web APIs: City of Toronto open data API

Public APIs vs Private APIs

- Public vs private refers to access, visibility, and documentation

Public APIs

- Available openly for any developer to use
- Well-documented and robustly coded to account for different (and untrusted) requests and input from the public
- Public Web APIs: Just need to sign up and get an API key to access
- e.g. GitHub, Spotify, YouTube

Public APIs vs Private APIs

Private APIs

- Access is restricted to internal apps or trusted external partners and requires authorization
- May be coded/documented for very specific use cases
- For interacting with internal data and functionality safely
- e.g.: APIs for internal tooling, bank partnerships

Public APIs vs Private APIs

- When writing APIs for public use:
  - Must validate inputs strictly
    - Defend against coding mistakes
    - Defend against malicious users (e.g. access to unauthorized data, system compromise)
  - Must document extensively
- Tradeoff between additional utility and engineering-hours

Web APIs

Why Web APIs?

- Your program can interact with the world
- Data from more than just files on the computer
  - Updated datasets
  - Industrial sensors
- Effects and outcomes on real-world objects
  - Smart home control
  - Mobile notifications

The RESTful Web API

- The RESTful Web APIs are a quasi-standard method of performing actions using or exchanging data with web-connected services
  - e.g. Retrieve list of repositories from GitHub
  - e.g. Using GPT-4 to process datasets automatically
  - e.g. Starting and stopping a container hosted on Microsoft Azure
- REST: “Representational State Transfer”
  - Uses HTTP requests: GET, POST, (PUT), (DELETE)
  - Generally, returns data in machine-readable formats like JSON

The RESTful Web API

- Uniform interface
  - Every entity (piece of data) is generally retrieved from the same URI
- Client-server decoupling
  - The web interface is assumed to be the only link between the client and the server
  - Data isn't getting passed through a separate channel (e.g. file on hard drive)
- Statelessness
  - All required information is included in the request
  - Identity is established on every request (usually via a secret token)

Web APIs

The RESTful Web API

Client

GitHub API
(https://api.github.com)

GET /user
Authorization Bearer zn4 … 2l3

*“Get details about myself.
I am identified by … “*

```
{ "login": "octocat", "id": 1,
"node_id": "MDQ6VXNlcjE=",
"avatar_url":
"https://github.com/images/error/oct
ocat_happy.gif", "gravatar_id": "",
"url":
"https://api.github.com/users/octoca
t", "html_url":
"https://github.com/octocat", … }
```

*“Here are your details…
.. beeepboop …”*

Web APIs

The RESTful Web API

Client

GitHub API
(https://api.github.com)

DELETE /repos/owner/thisrepo
Authorization Bearer zn4 … 2l3

“Delete this repository!
I am identified by … “

`Status: 204`

“Done!” 💥

API keys

- Unique string that acts like a password obtained by registering as a developer
  - Usually time-limited (hours to months)
- Identifies the client: Usage tracking and access limits
- API keys are usually sent in the request header
  - e.g. `Authorization: Bearer 2374823784282344 2`
- Keep your key secret - don't share or expose!
  - Store using secrets manager or protected configuration file (.gitignore is your friend!)

Web API responses

- Consists of an HTTP response code + body
- Response codes:
  - 2xx = success (e.g. 200)
  - 4xx = error with the request (e.g. 404 URI not found, 400 bad request)
  - 5xx = error with the server (e.g. 500 internal server error)
- The body is generally in JSON format (rarely, but sometimes in XML)

Web APIs in Python

Making Web API requests in Python

- Use the **`requests`** library to communicate with the API
- Use the **`json`** library to parse JSON responses

- Sometimes companies release Python libraries that make it easier to use their APIs from Python
  - Simplify authentication, request validation, response parsing, etc...

$> Interactive live coding

1. Generate a GitHub Personal Access Token (API key)

   Settings > Developer Settings > Personal access tokens

2. Add token to YAML file

This is a type of configuration!

3. Refer to GitHub API Documentation
4. Use **`requests`** to retrieve own user details from GitHub Web API

APIs in Python

$> Interactive live coding

```
import requests
import yaml
from pprint import pprint

with open('github_config.yml', 'r') as f:
    config = yaml.load(f)

response = requests.get(url='https://api.github.com/user',
                        headers={'Authorization': 'Bearer ' + config['token']})

# print raw response
print(response.status_code)
print(response.text)

# parse json
response_json = response.json()
pprint(response_json)

# print some values
print('Username: ' + response_json['login'])
print('Name: ' + response_json['name'])
```

$> Interactive live coding

1. Refer to ntfy.sh documentation
2. Subscribe to the dsi_c2_brs topic on your phone/computer
3. Send a message to the dsi_c2_brs topic

APIs in Python

$> Interactive live coding

```python
import requests

topic = 'dsi_c2_brs'
title = 'Hello, world!'
message = 'Hello, world from Simeon!'

# send a message through ntfy.sh
requests.post(
    'https://ntfy.sh/' + topic,
    data=message.encode('utf-8'),
    headers={'Title': title}
)
```

Exercise: Explore the GitHub API

- Using the GitHub API documentation:
  - choose an interesting endpoint that returns data
  - write a Python request to retrieve data from that endpoint
  - *bonus*: visualize or analyze it in some way
- Ideas:
  - List the top 20 most starred repositories using /search/repositories
  - List the top 20 most followed users using /search/users
    - Hint: try using q=stars:>1 or q=followers:>1

Homework

References

- Research Software Engineering with Python by Damien Irving, Kate Hertweck, Luke Johnston, Joel Ostblom, Charlotte Wickham, and Greg Wilson (https://merely-useful.tech/py-rse/config.html)