REST APIs

Pierre Formont

Agenda

- What are REST APIs
- Main methods to interact with APIs
- How to use in Python
- Exercises

API

- API = **A**`pplication` **P**`rogramming` **I**`nterface`
- set of definitions and protocols to build and integrate software
- contract between a provider and a user
- provider declares how its service can be used
- **interacting with any language / library / web service <=> using an API**

API examples

Unix

```
cd ~
ls -lah .
cat .zshrc
```

Base Python

```
import csv
with open("resources/csv/users.csv", "r") as f:
    reader = csv.reader(f)
```

Python library

```
import pandas
data = pandas.read_csv(file_path)
```

All these code samples is based on the API the language or library is declaring.

RESTful APIs

APIs that conform to the REST (**RE**`presentational` **S**`tate` **T**`ransfer`) architecture style.

- **not a protocol or a standard** but a set of 6 architectural constraints
- designed for network architectures, especially client-server applications
- more lightweight than other alternatives, *i.e.* `SOAP`
- widely used in the industry, especially in microservices architecture where components are independent from each other but still need to interact often

REST constraints

1. **Client–server architecture**: the client does not need to know the internals of the server
2. **Cacheability**: whenever possible, responses should be cached to improve response time
3. **Statelessness**: each request is separate, unrelated from previous ones
4. **Layered system**: the client does not need to know if it's connected to the server directly or through intermediaries
5. **Code-on-demand**: the server can send executable code to the client if needed

REST constraints: uniform interface

- requested resources are identifiable and **separate from the representations** sent to the client (*e.g.* `JSON` data returned from a database query)
- resource representations contain enough information to manipulate resources
- messages returned to the client are self-descriptive
- Hypermedia as the engine of application state: after accessing a resource, the client should be able to use hyperlinks to find all other currently available actions they can take.

REST standards

As mentioned, REST is an architecture, not a standard.

However, standards are good for interoperability, reproducibility, ease-of-use, friction reduction, etc.

A set of base practices has been widely adopted for RESTful APIs:

- building atop HTTP methods (also called verbs): `GET`, `POST`, `DELETE`, etc.
- identify resources with nouns, *e.g.* if a resouce is dealing with animals, it should be called `animals`:
  - `GET /api/animals`: retrieve a list of animals
  - `POST /api/animals`: add a new animal
- use a consistent structured data format for responses: `json` or `xml`

A industry-wide effort to provide a standard has been started several years ago: the OpenAPI specification.

How to use REST APIs

4 components needed:

- *endpoint*: the URL of a resource on a server, *e.g.* `https://api.github.com/users/<username>`
- *method*: the HTTP method/verb to query this resource with, *e.g.* `GET`, `POST`, `PUT`, `DELETE` for the 4 most common methods
- *body* / *payload*: actual data passed to/from the server when getting data or performing an action, *e.g.* when creating a new resource, the body contains the necessary information to create the resource
- *headers*: additional information passed between the client and the server, *e.g.* authentication tokens

Note: not all endpoints support all methods, some are only to retrieve data, some only to add or update data, some can do all operations

REST operations: GET

- used to **retrieve** a resource
- the method used in a browser to display a webpage
- the URL can have parameters or not:
  - `https://api.github.com/user` has no parameters
  - `https://api.github.com/users/<username>` has a `username` parameter

REST operations: GET examples (1/2)

Github API: list users

```
curl \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>"\
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/users
```

REST operations: GET examples (1/2)

Github API: list users

```
curl \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>"\
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/users
```

`curl`: command line tool to send HTTP requests

REST operations: GET examples (1/2)

Github API: list users

```
curl \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>"\
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/users
```

Header that indicates that the client understands json

REST operations: GET examples (1/2)

Github API: list users

```
curl \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>"\
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/users
```

Header containing the authentication token to the Github API

REST operations: GET examples (1/2)

Github API: list users

```
curl \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>"\
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/users
```

Proprietary header (X–) that indicates the version of the Github API to use

REST operations: GET examples (1/2)

Github API: list users

```
curl \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>"\
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/users
```

Endpoint to request

REST operations: GET examples (1/2)

Github API: list users

```
curl \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>"\
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/users
```

returns


```
[
  {
    "login": "octocat",
    "id": 1,
    "node_id": "MDQ6VXNlcjE=",
    "avatar_url": "https://github.com/images/error/octocat_happy.gif",
    "gravatar_id": "",
    "url": "https://api.github.com/users/octocat",
    "html_url": "https://github.com/octocat",
    "followers_url": "https://api.github.com/users/octocat/followers",
    "following_url": "https://api.github.com/users/octocat/following{/other_user}",
    "gists_url": "https://api.github.com/users/octocat/gists{/gist_id}",
    "starred_url": "https://api.github.com/users/octocat/starred{/owner}{/repo}",
    "subscriptions_url": "https://api.github.com/users/octocat/subscriptions",
    "organizations_url": "https://api.github.com/users/octocat/orgs",
    "repos_url": "https://api.github.com/users/octocat/repos",
    "events_url": "https://api.github.com/users/octocat/events{/privacy}",
    "received_events_url": "https://api.github.com/users/octocat/received_events",
    "type": "User",
    "site_admin": false
```

REST operations: GET examples (2/2)

Github API: retrieve a specific user

```
curl \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>"\
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/users/USERNAME
```

returns


```
{
  "login": "octocat",
  "id": 1,
  "node_id": "MDQ6VXNlcjE=",
  "avatar_url": "https://github.com/images/error/octocat_happy.gif",
  "gravatar_id": "",
  "url": "https://api.github.com/users/octocat",
  "html_url": "https://github.com/octocat",
  "followers_url": "https://api.github.com/users/octocat/followers",
  "following_url": "https://api.github.com/users/octocat/following{/other_user}",
  "gists_url": "https://api.github.com/users/octocat/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/octocat/starred{/owner}{/repo}",
  "subscriptions_url": "https://api.github.com/users/octocat/subscriptions",
  "organizations_url": "https://api.github.com/users/octocat/orgs",
  "repos_url": "https://api.github.com/users/octocat/repos",
  "events_url": "https://api.github.com/users/octocat/events{/privacy}",
  "received_events_url": "https://api.github.com/users/octocat/received_events",
  "type": "User",
  "site_admin": false,
  "name": "monalisa octocat",
  "company": "GitHub",
```

```
  "blog": "https://github.com/blog",
  "location": "San Francisco",
  "email": "octocat@github.com",
  "hireable": false,
  "bio": "There once was..."
```

REST operations: POST

- used to **add** a resource
- the URL can have parameters or not
- the necessary data should be put in the request *body*

REST operations: POST example

Github API: create a repository for the authenticated user

```
curl \
  -X POST \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>"\
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/user/repos \
  -d '{"name":"Hello-World","description":"This is your first repo!","homepage":"https://github.com","private
```

REST operations: POST example

Github API: create a repository for the authenticated user

```
curl \
  -X POST \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>"\
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/user/repos \
  -d '{"name":"Hello-World","description":"This is your first repo!","homepage":"https://github.com","private
```

HTTP method to use – here `POST`.

Not specified in the previous examples as `GET` is the default method.

REST operations: POST example

Github API: create a repository for the authenticated user

```
curl \
  -X POST \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>"\
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/user/repos \
  -d '{"name":"Hello-World","description":"This is your first repo!","homepage":"https://github.com","private
```

Body of the request

REST operations: POST example

Github API: create a repository for the authenticated user

```
curl \
  -X POST \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>"\
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/user/repos \
  -d '{"name":"Hello-World","description":"This is your first repo!","homepage":"https://github.com","private
```

returns


```
{
  "id": 1296269,
  "node_id": "MDEwOlJlcG9zaXRvcnkxMjk2MjY5",
  "name": "Hello-World",
  "full_name": "octocat/Hello-World",
  "owner": {
    "login": "octocat",
    "id": 1,
    "node_id": "MDQ6VXNlcjE=",
    "avatar_url": "https://github.com/images/error/octocat_happy.gif",
    "gravatar_id": "",
    "url": "https://api.github.com/users/octocat",
    "html_url": "https://github.com/octocat",
    "followers_url": "https://api.github.com/users/octocat/followers",
    "following_url": "https://api.github.com/users/octocat/following{/other_user}",
    "gists_url": "https://api.github.com/users/octocat/gists{/gist_id}",
    "starred_url": "https://api.github.com/users/octocat/starred{/owner}{/repo}",
    "subscriptions_url": "https://api.github.com/users/octocat/subscriptions",
    "organizations_url": "https://api.github.com/users/octocat/orgs",
```

```
    "repos_url": "https://api.github.com/users/octocat/repos",
    "events_url": "https://api.github.com/users/octocat/events{/privacy}",
    "received_events_url": "https://api.github.com/users/octocat/received_events",
    "type": "User",
    "site_admin": false
  },
  "private": false
```

REST operations: PUT

- used to **update** a resource
- the URL can have parameters or not
- the necessary data should be put in the request *body*

REST operations: PUT example

Github API: add an organization project to a team

```
curl \
  -X PUT \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>"\
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/orgs/ORG/teams/TEAM_SLUG/projects/PROJECT_ID \
  -d '{"permission":"write"}'
```

REST operations: DELETE

- used to **delete** a resource
- the URL can have parameters or not

REST operations: DELETE example

Github API: delete a team

```
curl \
  -X DELETE \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>"\
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/orgs/ORG/teams/TEAM_SLUG
```

Status codes

- code with 3 numbers returned by the API after each request
- indicates if an operation has succeeded or not
- 5 main families:
  - 100-199: information
  - 200-299: successful response, *e.g.* `200 OK`, `201 Created`
  - 300-399: redireciton, *e.g.* `301 Moved Permanently`
  - 400-499: client error, *e.g.* `401 Unauthorized`, `404 Not Found`
  - 500-599: server error, *e.g.* `500 Internal Server Error`

Recap

- architecture style for network client-server applications
- allow resource manipulation without knowledge of server internals
- need an endpoint and optional headers and body to communicate
- 4 main methods: `GET`, `POST`, `PUT`, `DELETE`
- status code in the response to indicate the result of the operation

REST APIs in Python

Python provides an HTTP client through the urllib.request module.

However, the recommended way – even by the Python maintainers – is through the requests package.

```
# make sure the virtual environment is activated
pip install requests
```

We can start testing the Github API using `requests`:

```
import requests
r = requests.get('https://api.github.com/events')
r
```

```
<Response [200]>
```

We can print the full response

```
[{'id': '26340783645',
  'type': 'CreateEvent',
  'actor': {'id': 49699333,
   'login': 'dependabot[bot]',
   'display_login': 'dependabot',
   'gravatar_id': '',
   'url': 'https://api.github.com/users/dependabot[bot]',
   'avatar_url': 'https://avatars.githubusercontent.com/u/49699333?'},
  'repo': {'id': 587409988,
   'name': 'gertnerbot/dev_project',
   'url': 'https://api.github.com/repos/gertnerbot/dev_project'},
  'payload': {'ref': 'dependabot/pip/sphinx-6.1.3',
```

```
   'ref_type': 'branch',
   'master_branch': 'master',
   'description': None,
   'pusher_type': 'user'},
  'public': True,
  'created_at': '2023-01-10T17:29:44Z'},
 {'id': '26340783590',
  'type': 'PushEvent',
  'actor': {'id': 2168535,
```

REST APIs in Python - parameters

Parameters are passed as Python dictionaries to the `params` argument:

```
1  params = {'per_page': 5}
2  r = requests.get('https://api.github.com/events', params=params)
3  r.json()
```


```
[{'id': '26340784086',
  'type': 'IssueCommentEvent',
  'actor': {'id': 622269,
   'login': 'vusters',
   'display_login': 'vusters',
   'gravatar_id': '',
   'url': 'https://api.github.com/users/vusters',
   'avatar_url': 'https://avatars.githubusercontent.com/u/622269?'},
  'repo': {'id': 468481476,
   'name': 'EnsembleUI/ensemble',
   'url': 'https://api.github.com/repos/EnsembleUI/ensemble'},
  'payload': {'action': 'created',
   'issue': {'url': 'https://api.github.com/repos/EnsembleUI/ensemble/issues/67',
    'repository_url': 'https://api.github.com/repos/EnsembleUI/ensemble',
    'labels_url': 'https://api.github.com/repos/EnsembleUI/ensemble/issues/67/labels{/name}',
    'comments_url': 'https://api.github.com/repos/EnsembleUI/ensemble/issues/67/comments',
    'events_url': 'https://api.github.com/repos/EnsembleUI/ensemble/issues/67/events',
    'html_url': 'https://github.com/EnsembleUI/ensemble/issues/67',
    'id': 1434943330,
    'node_id': 'I_kwDOG-x1xM5Vh39i',
    'number': 67,
    'title': 'Ability to define environment variables',
```


Check that there are only 5 results:

```
1  len(r.json())
```

REST APIs in Python - headers

Headers are also passed as Python dictionaries – to the `headers` argument:

```
1 headers = {'accept': 'application/vnd.github+json'}
2 r = requests.get('https://api.github.com/events', headers=headers)
3 r.json()
```


```
[{'id': '26340784086',
  'type': 'IssueCommentEvent',
  'actor': {'id': 622269,
   'login': 'vusters',
   'display_login': 'vusters',
   'gravatar_id': '',
   'url': 'https://api.github.com/users/vusters',
   'avatar_url': 'https://avatars.githubusercontent.com/u/622269?'},
  'repo': {'id': 468481476,
   'name': 'EnsembleUI/ensemble',
   'url': 'https://api.github.com/repos/EnsembleUI/ensemble'},
  'payload': {'action': 'created',
   'issue': {'url': 'https://api.github.com/repos/EnsembleUI/ensemble/issues/67',
    'repository_url': 'https://api.github.com/repos/EnsembleUI/ensemble',
    'labels_url': 'https://api.github.com/repos/EnsembleUI/ensemble/issues/67/labels{/name}',
    'comments_url': 'https://api.github.com/repos/EnsembleUI/ensemble/issues/67/comments',
    'events_url': 'https://api.github.com/repos/EnsembleUI/ensemble/issues/67/events',
    'html_url': 'https://github.com/EnsembleUI/ensemble/issues/67',
    'id': 1434943330,
    'node_id': 'I_kwDOG-x1xM5Vh39i',
    'number': 67,
    'title': 'Ability to define environment variables',
```

REST APIs in Python - body

Body is also passed as a Python dictionary – to the `data` argument:

```
data = {'param1': 'value1'}
r = requests.post('https://api.github.com/events', data=data)
```

Note: this sends the data as form-encoded. Some APIs (including Github's) accepts json-encoded payloads. In that case, use the `json` parameter:

```
data = {'param1': 'value1'}
r = requests.post('https://api.github.com/events', json=data)
```

REST APIs in Python - basic exercises

Exercise 8

> Send a GET request to `https://httpbin.org/get` with an *'accept: application/json'* header. What is the response status code ?

Exercise 9

> Send a POST request to `https://httpbin.org/get`. What is the response status code and why ?

Exercise 10

> Send a POST request to `https://httpbin.org/post` using the `data` parameter. Send one using the `json` parameter. Note the differences.

OpenSky API

OpenSky is a non-profit aviation research network.

They provide a free REST API with information about flights.

Exercise 11

> Get all flights leaving the Paris Charles-de-Gaulle airport on 2022-12-01.

- Which endpoint do you need to use ?
- What are the parameters for this endpoint?
- In what format do you need to input them ?

> Find the airport with the most flights out of Paris Charles-de-Gaulle.

Github API - authentication

Most of the endpoints for the Github API require authentication.

You need to create a Personal Access Token from your Github settings and send it in the header of the request.

Exercise 12

> List all your repositories using the Github API

Check the documentation for how to format the headers.