Working with Web APIs (continued)

Lecture 15

(De)serialization in Ruby

- Get JSON from an object

```
JSON.generate ([0x10, true, :age, 'hi'])
#=> "[16,true,\"age\",\"hi\"]"
```

- Get an object from JSON

```
s = "{\"zips\": [43210, 43211]}"
JSON.parse(s)
#=> {'zips' => [43210, 43211]}
JSON.parse(s, symbolize_names: true)
#=> {:zips => [43210, 43211]}
```

Alternatives

- JSON is readable
  - Sometimes used for configuration files
    - VSCode: .vscode/settings.json
    - .markdownlint.json, devcontainer.json,…
- But JSON isn't human-friendly
  - No comments
  - Visual clutter with lots of " marks
- Alternatives, when readability matters
  - YAML: yet another markup language
  - JSONC: adds comment, not universal

Web APIs

- API contains endpoints, each of which:
  - verb (GET or POST) and URL path
  - Accepted arguments
  - Returned value (typically JSON)
- Roughly equivalent to a method signature
- Many ways to call an endpoint
  - Command line: curl
  - Tool: VSCode extensions rest-client, Postman
  - Ruby client gem: Faraday, Net::HTTP, httpx
  - Client library provided by the service itself (octokit for GitHub, stripe-ruby for Stripe)

Example APIs

- Dad Jokes
  - https://icanhazdadjoke.com/api
- Canvas (ie Carmen)
  - https://canvas.instructure.com/doc/api/
- US National Weather Service
  - https://www.weather.gov/documentation/services-web-api
- US Census Bureau
  - https://www.census.gov/data/developers/data-sets.html
- GitHub
  - https://docs.github.com/en/rest
- And many, many more…
  - https://github.com/public-apis/public-apis

Demo: Calling an API

- Curl to dad jokes

```
$ curl \
https://icanhazdadjoke.com/search?term=computer
$ curl \
https://icanhazdadjoke.com/search?term=computer \
  -H "Accept: application/json"
```

- Browser to Carmen API

```
https://osu.instructure.com/api/v1/courses
```

- HTTPX gem to dad jokes

```
require 'httpx'
resp = HTTPX.get('https://icanhazdadjoke.com',
     headers: {'Accept' => 'application/json'})
puts resp.body
puts resp.json['joke']
```

API Key

- Service may require a key to use
  - Register with service, get a secret token (ie a long random number or string)
  - Include this token in every HTTP request, eg using the Authorization header

    ```
    Authorization: Bearer 8497~Xd0aaaaaIMadeThisUpzzzz
    ```
- Golden rule: never share or commit your secret token!
  - Treat it like a password
  - Dilemma: Your code needs to use it, so it needs to be stored somewhere…

Solution Strategy: Env Variable

- Keep .env file out of commits!

```
# .gitignore
.env
```

- Create .env file for secret(s)

```
# .env
CANVAS_TOKEN=YOUR_SECRET_VALUE
```

- Create sample with dummy value(s)

```
# .env.template
CANVAS_TOKEN=CANVAS_TOKEN_SECRET
```

- Use environment variable in client code

```
require 'dotenv'
Dotenv.load # looks for .env file
auth = "Bearer #{ENV['CANVAS_TOKEN']}"
req.header['Authorization'] = auth
```

Getting an API Key

- GitHub
  - Login, Settings > Developer Settings
  - Personal access tokens > Tokens
- Canvas
  - Login, Account > Settings
  - Under "Approved Integrations", "+ New Access Token"
- Use meaningful name for token
- Value typically shown just one time

Summary

- Passing arguments
  - GET: query string (url-encoded)
  - POST: body (several different encodings)
- JSON
  - Syntax for describing values
  - Just a few basic types (object, array, text, number…)
  - Useful for (de)serialization, while also human-readable
- API endpoints
  - Response body is often JSON
- API keys
  - Protect secrets, eg with private .env file
  - Use in request header to legitimize source