Data Acquisition via Application Programmable Interfaces

Adrian Petrescu

Kinaxis

2023-11-17

Introduction

Adrian Petrescu
Distinguished Architect at Kinaxis

Logistics

- GitHub Repository: https://github.com/apetresc/rotman-api
- Notebook: RAC Link

Schedule

Part 1 (Today)

- Introduction and motivation
- HTTP, REST, and the languages of the web
- Authentication schemes
- Lots and lots of practical examples

Part 2

- Scraping unstructured data from the web
- Parsing
- Automated spiders

Part 3

- The server-side of APIs
- Deploying to the Cloud
- Project Description

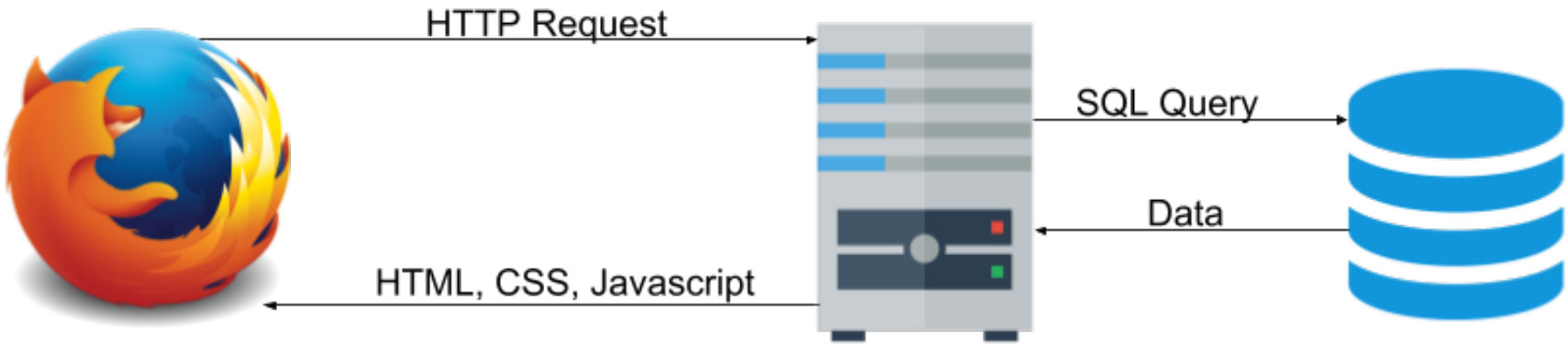
What is an API?

What is an API?

What is an API?

Web Architecture

In Its Simplest Possible Form



Web Architecture

In Its Modern Form

HTTP

Hyertext Transfer Protocol

- The language of the web
- Transactional
- Stateless

Anatomy of a Request

Marco

```
GET /api/v2/evolution-chain/1?format=json HTTP/1.1
```

Anatomy of a Request

Marco

```
GET /api/v2/evolution-chain/1?format=json HTTP/1.1

Host: pokeapi.co
Connection: keep-alive
Accept: application/json
```

Anatomy of a Response

Polo

```
HTTP/1.1 200 OK
Server: nginx
Content-Type: application/json
Content-Length: 3238
[...]


{
  "id": 1,
  "chain": {
    "species": "bulbasaur",
    [...]
  }
}
```

Methods

- `HEAD`
- `GET`
- `POST/PUT`
- `DELETE`

Return Codes

- 200 Success
- 300 Redirect
- 400 Client Error
- 500 Server Error

Return Codes — 200 Success

- `200 OK`
- `201 Created`
- `202 Accepted`
- `204 No Content`
- `206 Partial Content`

Return Codes — 300 Redirect

- `301 Moved Permanently`
- `302 Moved Temporarily`
- `304 Not Modified`

Return Codes — 400 Client Error

- `400 Bad Request`
- `401 Unauthorized`
- `403 Forbidden`
- `404 Not Found`
- `405 Method Not Allowed`

Return Codes — 500 Server Error

- `500 Internal Server Error`
- `501 Not Implemented`
- `503 Service Unavailable`

Some Important Headers

- `Set-Cookie` / `Cookie`
- `User-Agent`
- `Expires` / `Cache-Control`

A much more comprehensive listing can be found on the Mozilla Developer Network.

JSON

JavaScript Object Notation

```
{
  "a": true,
  "b": {
    "c": 1,
    "d": [
      {"x": 1},
      {"y": 2}
    ]
  }
}
```

XML

eXtensible Markup Language

```
<card xmlns="http://businesscard.org/schema.xml">
  <name>John Doe</name>
  <title>CEO, Widget Inc.</title>
  <email>john.doe@widget.com</email>
  <phone>(202) 456-1414</phone>
  <logo url="widget.gif"/>
</card>
```

XSD

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:b="http://businesscard.org"
        targetNamespace="http://businesscard.org">

  <element name="card" type="b:card_type"/>
  <element name="name" type="string"/>
  <element name="title" type="string"/>
  <element name="email" type="string"/>
  <element name="phone" type="string"/>
  <element name="logo" type="b:logo_type"/>

  <complexType name="card_type">
    <sequence>
      <element ref="b:name"/>
      <element ref="b:title"/>
      <element ref="b:email"/>
      <element ref="b:phone" minOccurs="0"/>
      <element ref="b:logo" minOccurs="0"/>
    </sequence>
  </complexType>

  <complexType name="logo_type">
    <attribute name="url" type="anyURI"/>
  </complexType>

</schema>
```

Authentication and Authorization

- Unauthenticated
- HTTP Basic Auth
- API Access Tokens
- OAuth 1/2

HTTP Basic Auth

Example

```
GET /user HTTP/1.1

Host: api.github.com
Authorization: Basic YXBldHJlc2M6MFk4aWJYdU7wvabj
```

- Extremely easy to implement
- This does authentication, not authorization, in spite of the header
- Not secure - the string there is just a base64 encoding, not any sort of encryption

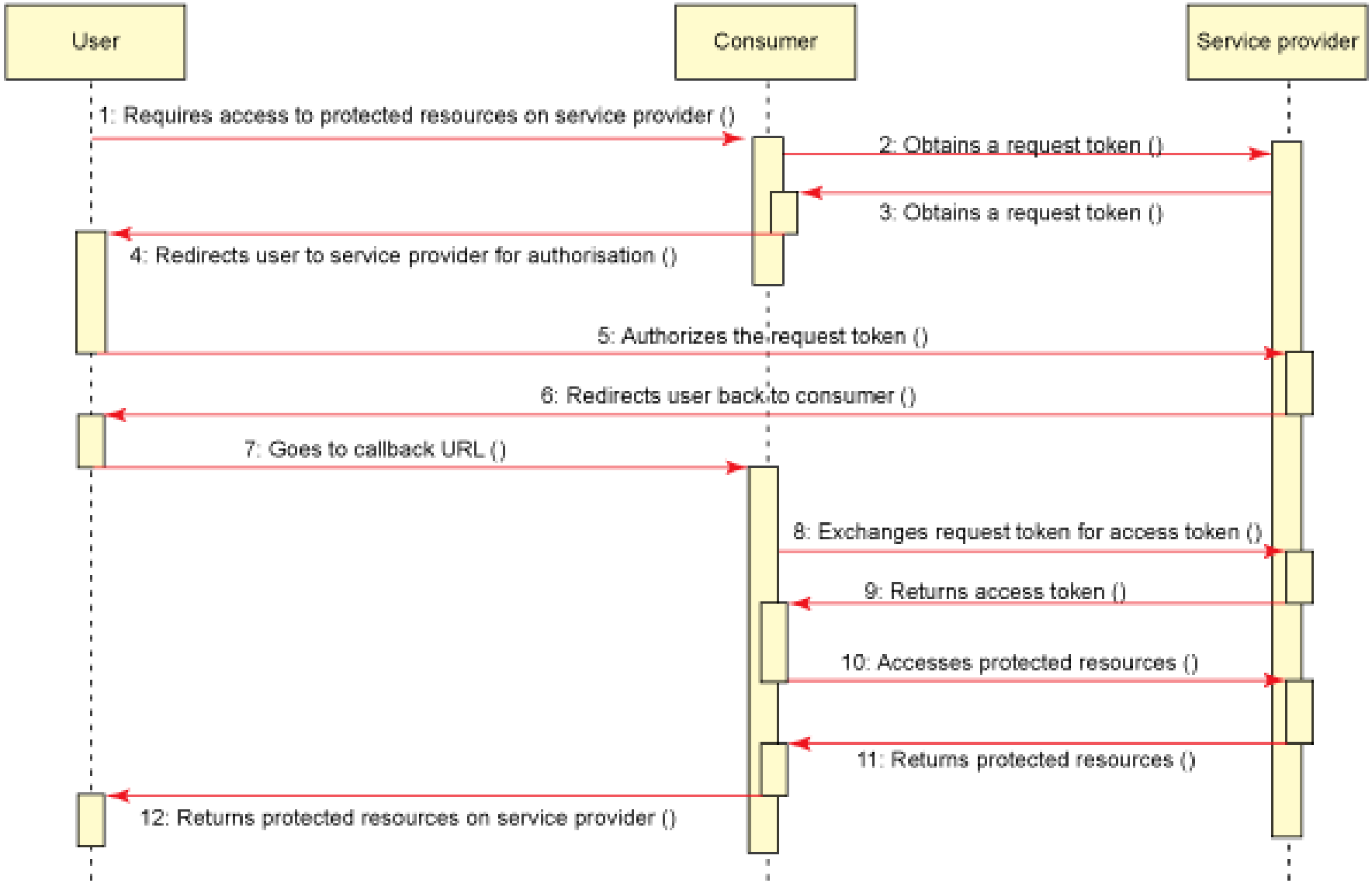
API Access Token

Example

```
GET /v3/businesses/q9_gLvTNf11etVxbH7JYOQ HTTP/2
Host: api.yelp.com
Authorization: Bearer p1hiON8SYR-Z[...]
```

- Easy to use for clients
- Only works for server-side, single-user applications

OAuth


User
Consumer
Service provider
1: Requires access to protected resources on service provider ()
2: Obtains a request token ()
3: Obtains a request token ()
4: Redirects user to service provider for authorisation ()
5: Authorizes the request token ()
6: Redirects user back to consumer ()
7: Goes to callback URL ()
8: Exchanges request token for access token ()
9: Returns access token ()
10: Accesses protected resources ()
11: Returns protected resources ()
12: Returns protected resources on service provider ()

Luckily, this whole dance can be automated away with the right library!

Request a token

```
from requests_oauth2.services import GoogleClient
google_auth = GoogleClient(
      client_id="your-google-client-id",
      client_secret="super-secret",
      redirect_uri="http://localhost:5000/google/oauth2callback",
      )

    authorization_url = google_auth.authorize_url(
        scope=["email"],
        response_type="code",
        )
```

Authorize with the token

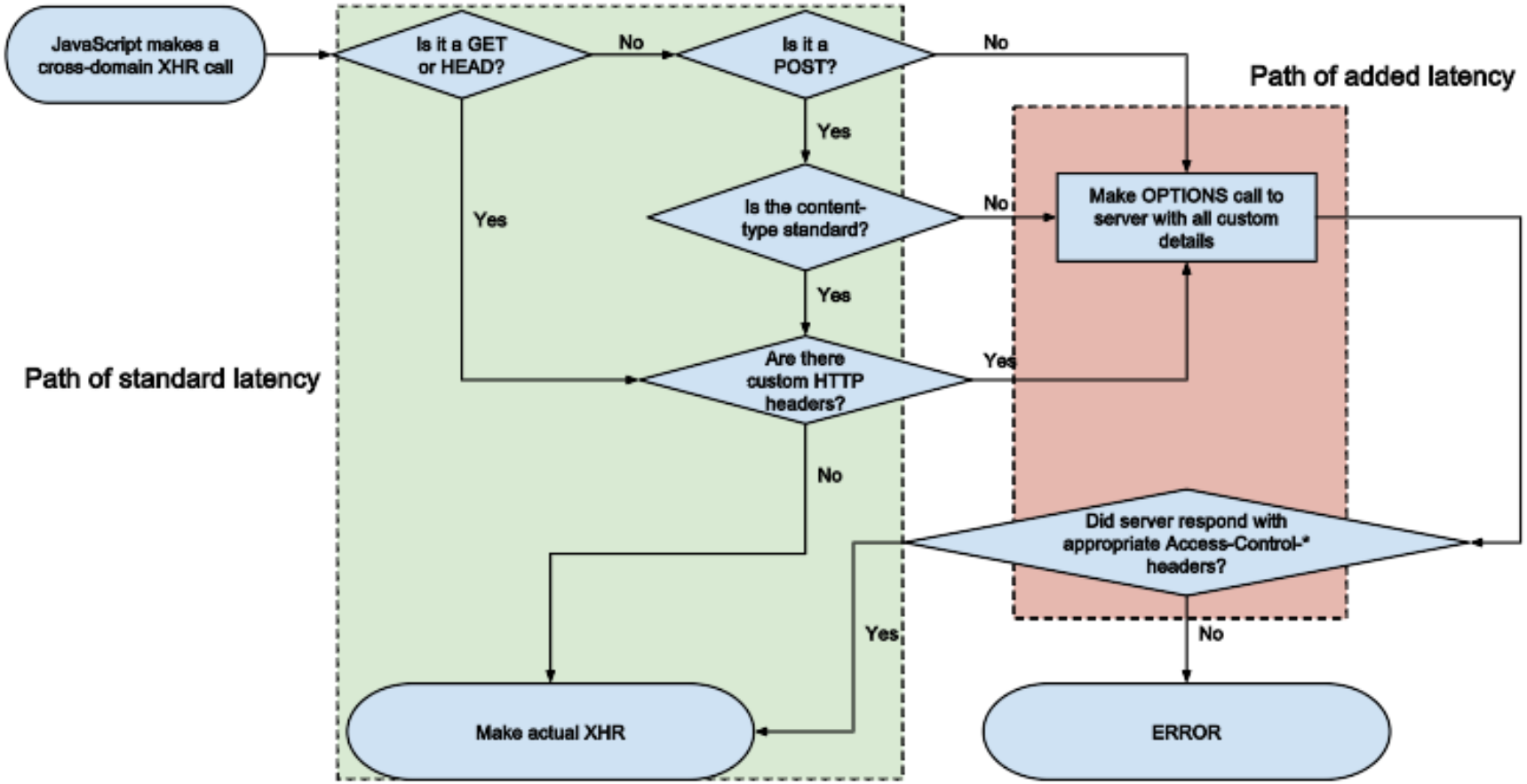
```
code = get_request_parameter("code")
data = google_auth.get_token(
      code=code,
      grant_type="authorization_code",
      )

session["access_token"] = data["access_token"]

with requests.Session() as s:
    s.auth = OAuth2BearerToken(access_token)
    r = s.get("https://www.googleapis.com/plus/v1/people/me")
    r.raise_for_status()
    data = r.json()
```

CORS

Cross-Origin Resource Sharing



Demo - GitHub

Demo - Yelp