REST API Introduction

Topics

1) How to request and send data to a server?
2) How to design a server's API?

HTTP

Overview

- **Front-end** = client-side; browser

- **Back-end** = server side

- **Why make web-based app?**
  - server to allow interaction between users
  - server to store resources or do heavy processing
  - centrally managed deployment and admin

Server Interaction

- Browser getting data from webserver
  - browser does HTTP GET on URL
  - server sends back a web page (HTML, CSS, JS)

- Font-end/Back-end Interaction
  - client-side makes requests to server's RESTful API's endpoints (URLS)
  - data transmitted in JSON (or XML)

HTTP

- HTTP:..

- URL:..
  - Ex: http://www.sfu.ca/~bfraser/answers

    <protocol>://<domain name>/<path>

    <protocol>://<domain name>:<port>/<path>

- Protocol ports
  - HTTP: 80 (or 8080 alt)
  - HTTPS: 443 (or 8443 alt)
           S = Secure

HTTP Methods

- HTTP methods:
  What does the client want to happen at a URL?

- These are the..
  - : retrieve some information from the URL: does not change server state
  - : Submit a new entity (object) to the URL
  - : Delete some entity (object) at the URL
  - : Replace an entity at the URL with new value
  - ... omitting HEAD, CONNECT, OPTIONS, TRACE, PATCH

HTTP Response Status Codes

- Each request message (a GET, POST, ...) returns a response code:
    - 200:..
    - 201:..
    - 400:   Bad Request (client-side error)
    - 401:   Unauthorized (who are you?)
    - 403:   Forbidden (I know who you are, but still not allowed)
    - 404:..
    - 500:   Server-side error
    - (... many omitted!)

Sending Data to the Server

- Front end can send data to the server via:
    - : Put data in path variables
        - Ex: GET http://my.com/api/person/5
    - : for GET only;
      no raw special characters (Ex: %20 = space)
        - Ex: https://www.google.com/search?q=hi+world
    - : All HTTP messages have header
        - Ex: authentication or apiKey
          "ApiKey:abc123"
    - : Block of data (often text such as JSON)
        - Ex: {"name":"Dr. Evil","age":95,"laugh":"Mwahah"}

URL Path Variables Details

- Path Variable Idea
  - URL encodes groups or categories as though they are "folders", and items as "files"

- Example
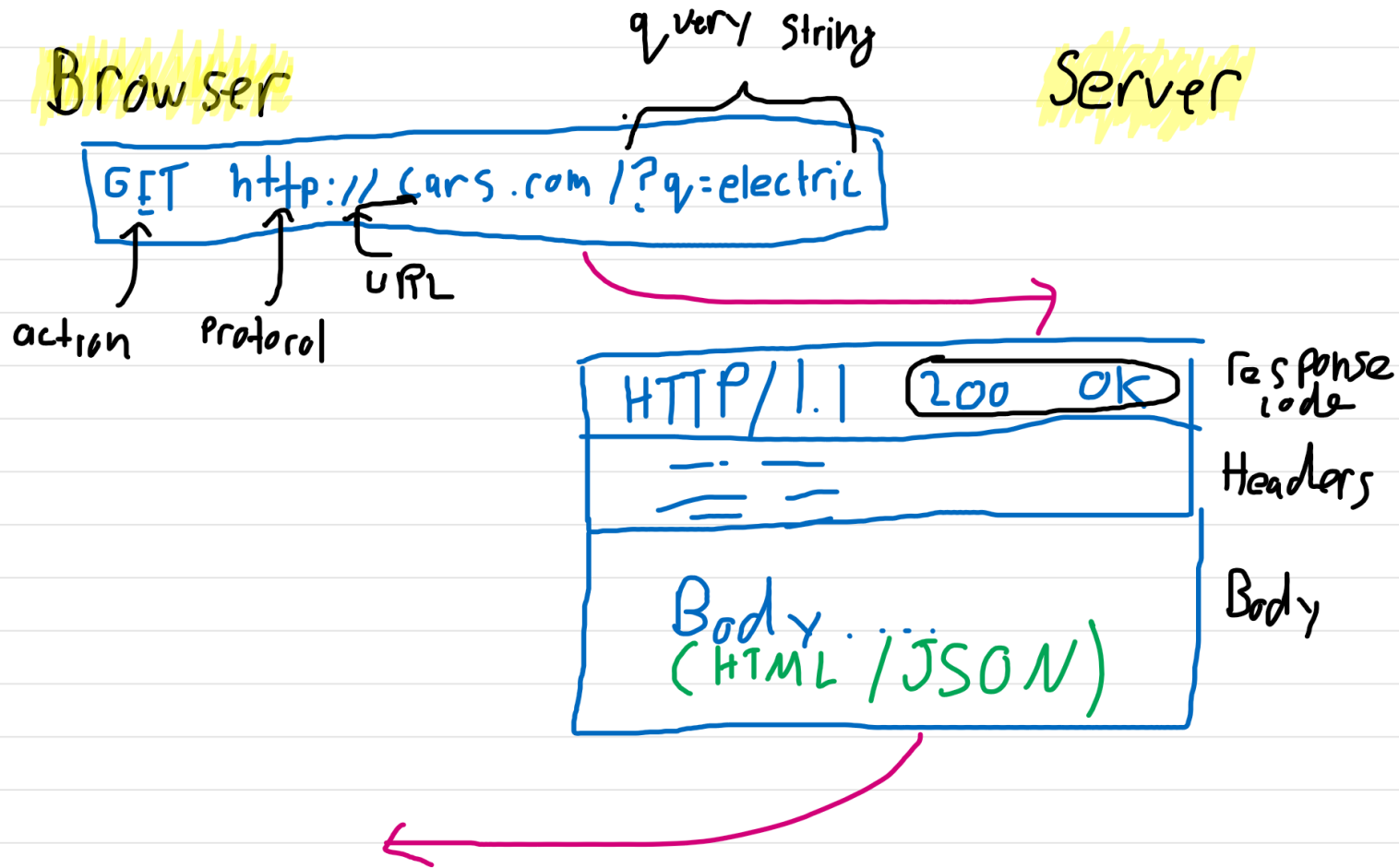 https://coursys.sfu.ca/2050sp-cmpt-276-d1/students/hiwId
  - It seems like we are browsing into folders for a specific file

  - ..

Query String Details

- Query String: the common way to send data for GET
  - Use to encode..
    - Ex: search queries

- Common Format
  http://my.com/s?key=value&otherkey=othervalue

- Demo
  curl -k -i -X GET https://www.adafruit.com/?q=wire

HTTP Body details

- HTTP messages can include a body
  - Used by POST and PUT to send data
  - Often a JSON structure or binary data
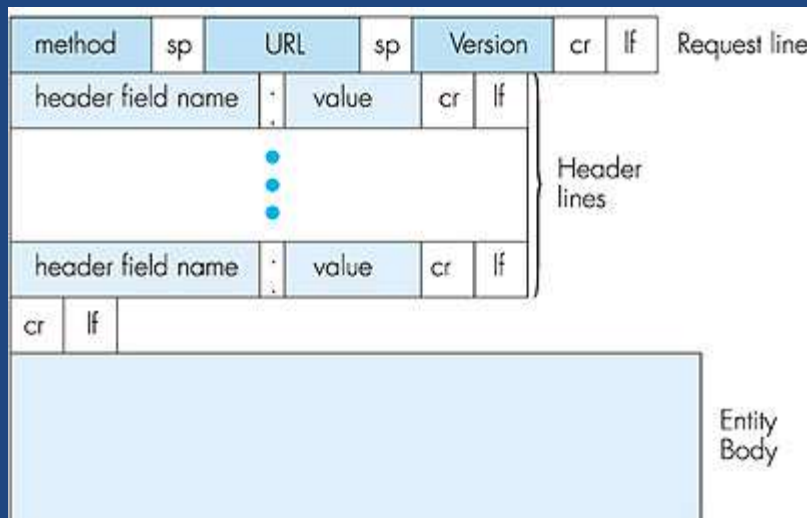
```
GET   /~bfraser/   HTTP/1.1
Host: www.sfu.ca
Connection: keep-alive
Cache-Control: no-cache
User-Agent: Mozilla/5.0 ...
Accept: text/html,application/...
```

```
HTTP/1.1 200 OK
Date: Mon, 02 Mar 2020 05:10:18 GMT
Server: Apache
box: b3 D=1361386 t=1583125818662494
Access-Control-Allow-Origin: *
Content-Length: 3795
Content-Type: text/html;charset=ISO-8859-1

<!DOCTYPE
<html>
 <head>
  <title>Index of /~bfraser</title>...
```

Image: http://netlab.ulusofona.pt/rc/book/2-application/2_02/index.htm

REST API

API & REST

- API:..
  - How a program exposes its functionality for other programs to use.

- REST:..
  - ..

  - It works with HTTP caching and semantics to improve performance
  - REST is founded on some principles, not a strict prescription.
    So what is "RESTful" is up to interpretation

- TLA: Three Letter Acronym

REST Example

- Example: Tic-tac-toe game
    - Base URL: my.com
    - /games   GET (list), POST (new)
    - /games/52   GET (info), POST (change info)
    - /games/52/moves   GET (list), POST (new)
    - /games/52/moves/1   GET (info), POST (change info)

- Full Example
  GET my.com/games/52/moves/1
    - In games API, retrieve info on game #52's move #1

REST Example (cont)

- Get Game Info

curl -X GET localhost/games/52

HTTP/1.1 200 OK
{
    "id":   52
    "user1": "Brian",
    "user2": "AI3",
    "href":   "/games/52"
}

- Data Structure

```
struct {
    int id;
    string user1;
    string user2;
    string href;
}
```

..
Simple data structure to send data from back-end to front-end

Martin Fowler: https://martinfowler.com/eaaCatalog/dataTransferObject.html

- **Get Moves**

```
curl -X GET localhost/games/52/moves

HTTP/1.1 200 OK
[
    {
        "id":       2,
        "user":    "Brian",
        "row":     1,
        "col":     1
    },
    {
        "id":       6,
        "user":    "AI3",
        "row":     3,
        "col":     1
    }
]
```

- **Make a move**

```
curl -X POST -d {
    "user":    "Brian",
    "row":     3,
    "col":     3
} localhost/games/52/moves
```

RESTful API Design

- Design API around things and actions
  - Structure URL for the hierarchical nature of the data

- Things (nouns)
  - Data you want to expose

  - ..

- Actions (verbs)
  - C   POST (or PUT)
  - R   GET
  - U   POST (or PUT if you are updating the whole item at once, not just part).
  - D   DELETE

RESTful API Design (cont)

- GET (and PUT) must be idempotent:
  - ..

- POST is a catch all for doing anything.

- Properties of RESTful
  - Server returns self-descriptive resources
  - Server maintains nothing about state of the connection; everything comes from HTTP headers, etc
  - Cache as much as possible to reduce server load
  - <...omitted more...>

Summary

- HTTP
  - Protocol for accessing resources via URL's

- HTTP Methods
  - GET, POST, DELETE, PUT, etc.

- Data in URL, Query String, Header, Body

- REST
  - Design URLs for Hierarchical data
  - REST properties