Handling HTTP Requests

SWE 432, Fall 2019

Web Application Development

Quiz

Go to:
b.socrative.com, Click student login
Room name: SWE432
Student Name: Your G-number (Including the G)

Reminder: Survey can only be completed if you are in class. If you are not in class and do it you will be referred directly to the honor code board, no questions asked, no warning.

Review: Express

```
var express = require('express');
```

Import the module express

```
var app = express();
```

Create a new instance of express

```
var port = process.env.port || 3000;
```

Decide what port we want express to listen on

```
app.get('/', function (req, res) {
  res.send('Hello World!');
});
```

Create a *callback* for express to call when we have a "`get`" request to "`/`". That callback has access to the request (`req`) and response (`res`).

```
app.listen(port, function () {
  console.log('Example app listening on port' + port);
});
```

Tell our new instance of express to listen on `port`, and print to the console once it starts successfully

Review: Route parameters

- Named URL segments that capture values at specified location in URL
  - Stored into `req.params` object by name
- Example
  - Route path */users/**:userId**/books/**:bookId***
  - Request URL *http://localhost:3000/users/34/books/8989*
  - Resulting `req.params: { "userId": "34", "bookId": "8989" }`

```
app.get('/users/:userId/books/:bookId', function(req, res) {
  res.send(req.params);
});
```

Review: Making HTTP Requests

- May want to request data from other servers from backend
- Fetch
  - Makes an HTTP request, returns a Promise for a response
  - Part of standard library in browser, but need to install library to use in backend
- Installing:

```
npm install node-fetch --save
```

- Use:

```
const fetch = require('node-fetch');

fetch('https://github.com/')
    .then(res => res.text())
    .then(body => console.log(body));

var res = await fetch('https://github.com/');
```

https://www.npmjs.com/package/node-fetch

Today

- Design considerations in identifying resources
- REST
    - What is it?
    - Why use it?

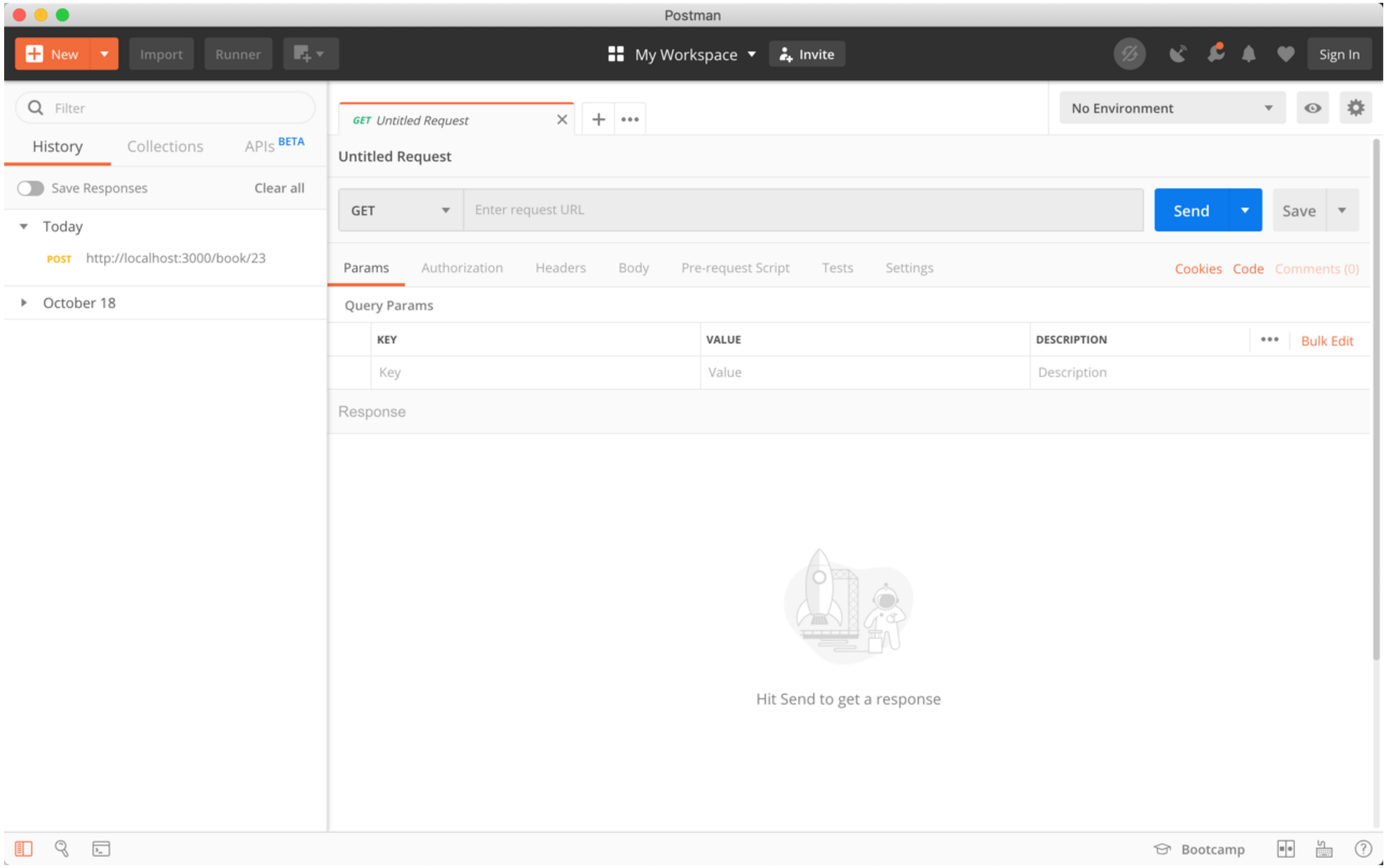
Logistics

- HW2 due on 10/7 (2 weeks)
- Questions

Demo: Using fetch to post data

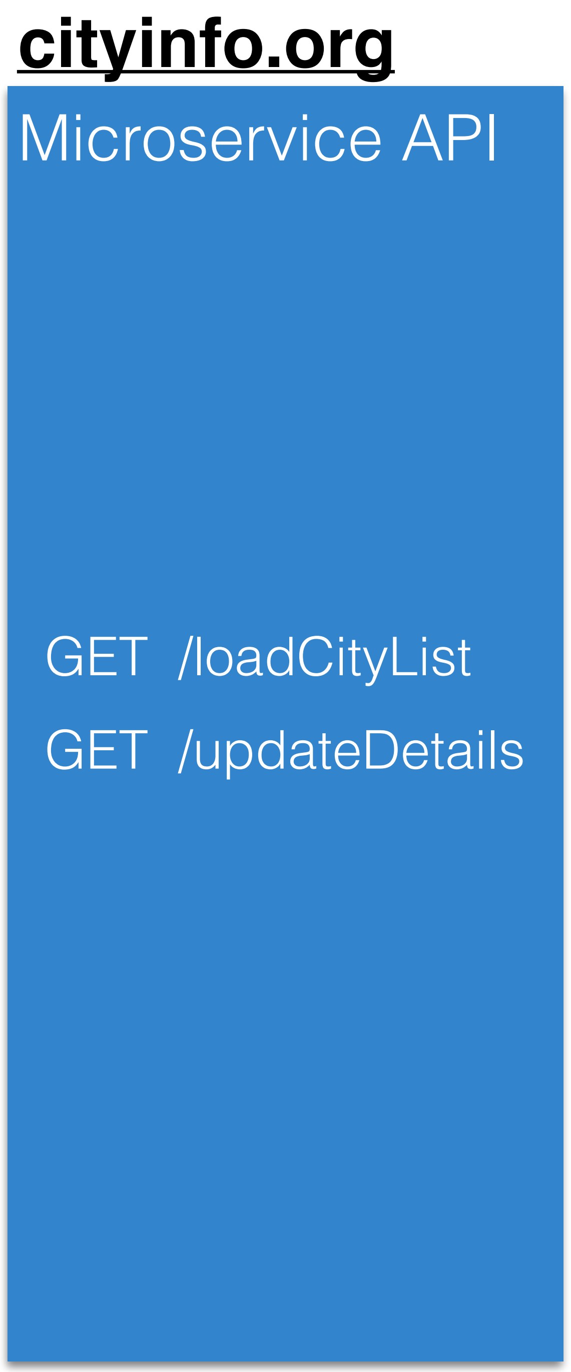
```
var express = require('express');
var app = express();
const fetch = require('node-fetch');

const body = { 'a': 1 };

fetch('http://localhost:3000/book/23', {
    method: 'post',
    body:    JSON.stringify(body),
    headers: { 'Content-Type': 'application/json' },
})
    .then(res => res.json())
    .then(json => console.log(json));
```

Demo: Making http request with postman

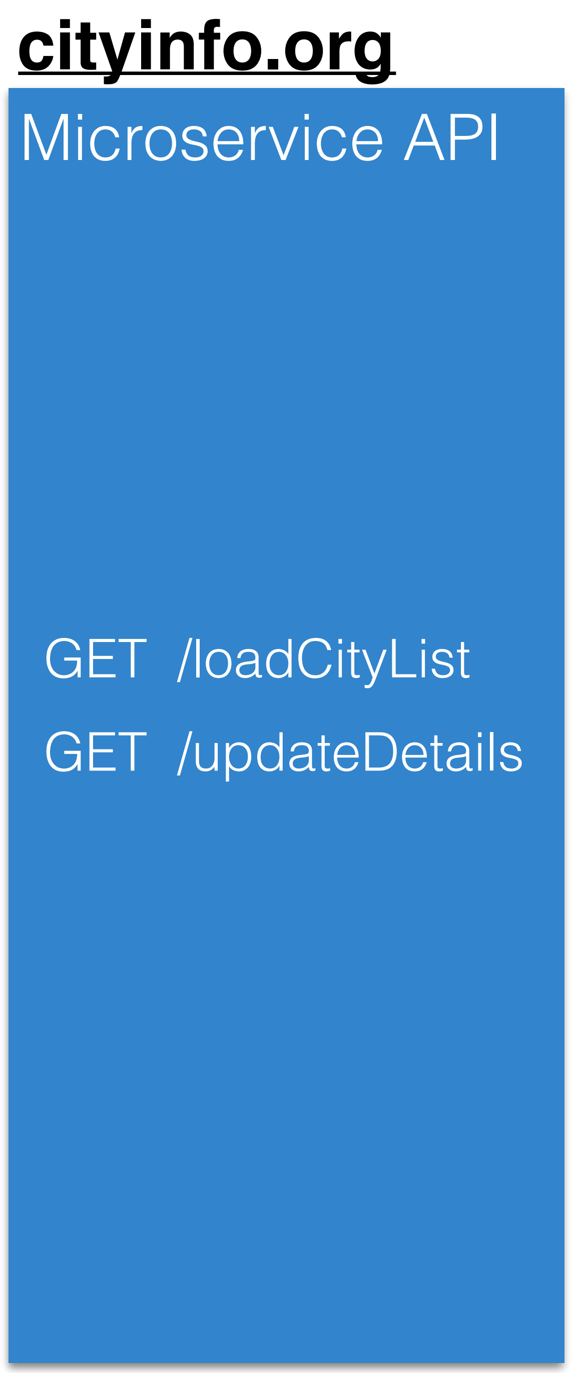


https://www.getpostman.com/

Demo: Building a microservice w/ Express



API: Application Programming Interface



- Microservice offers public **interface** for interacting with backend
  - Offers abstraction that hides implementation details
  - Set of endpoints exposed on micro service

- Users of API might include
  - Frontend of your app
  - Frontend of other apps using your backend
  - Other servers using your service

APIs for functions and classes

V1

```
function sort(elements)
{
   [sort algorithm A]
}
```

```
class Graph
{
   [rep of Graph A]
}
```

↓

Implementation change

Consistent interface

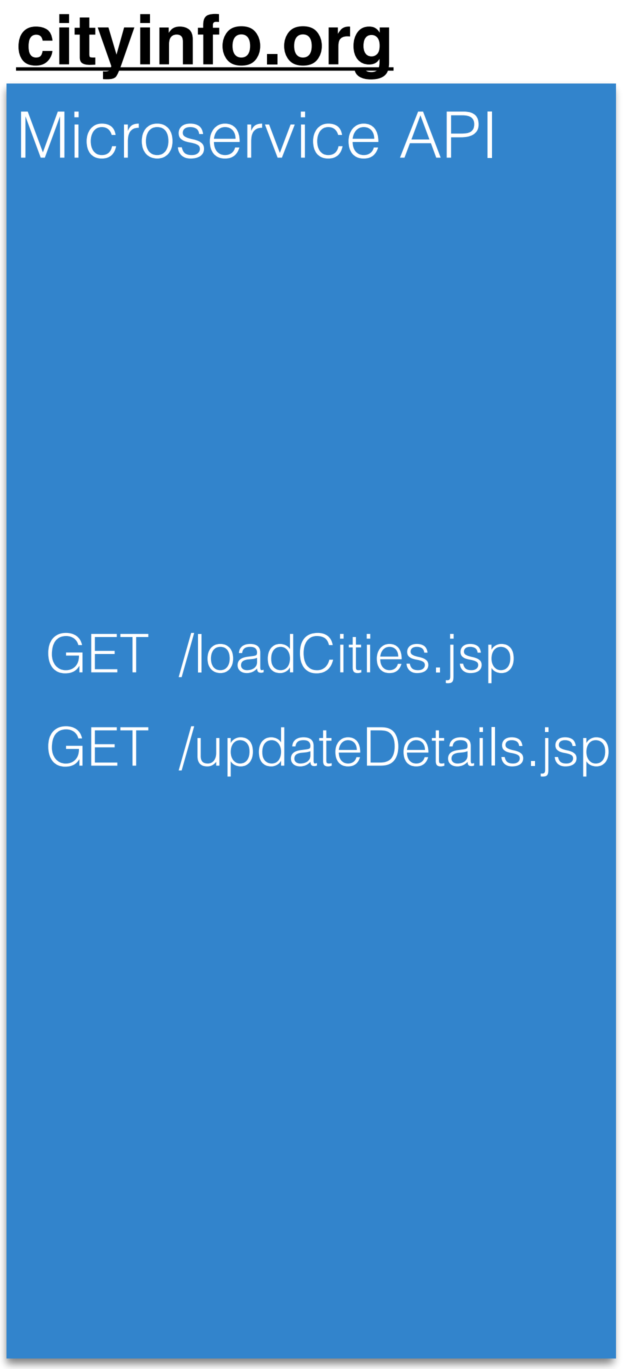
V2

```
function sort(elements)
{
   [sort algorithm B]
}
```

```
class Graph
{
   [rep of Graph B]
}
```

Support scaling

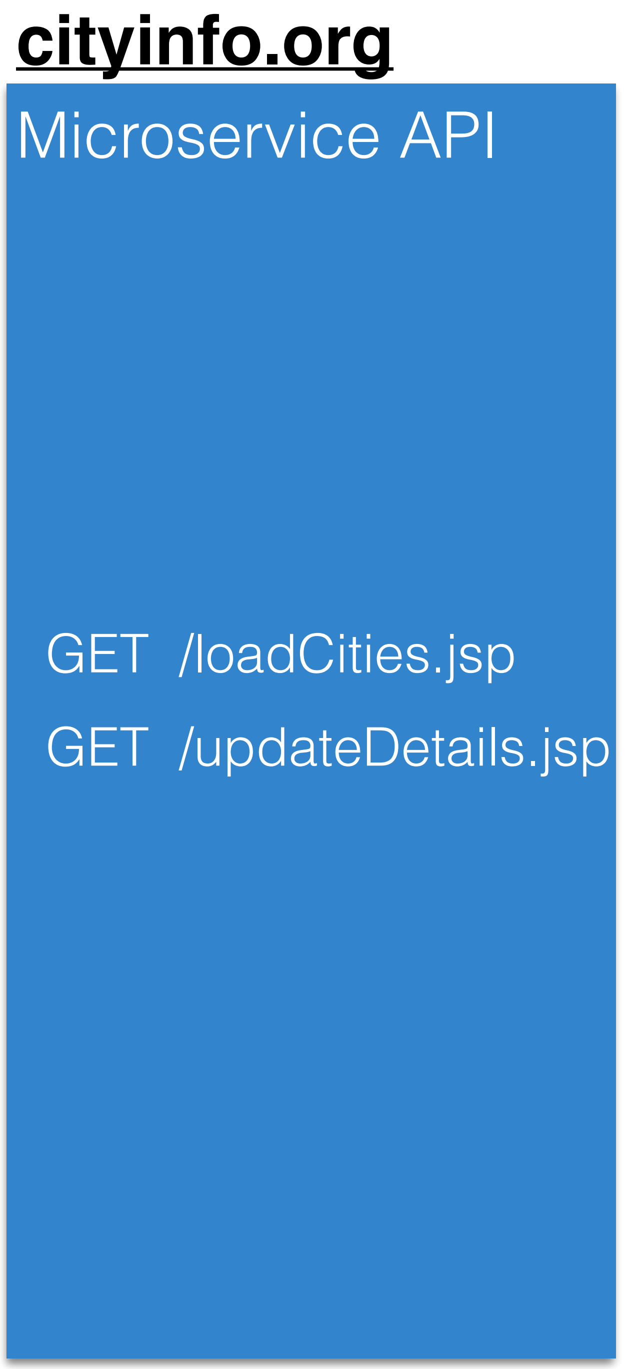
- Yesterday, cityinfo.org had 10 daily active users. Today, it was featured on several news sites and has 10,000 daily active users.
- Yesterday, you were running on a single server. Today, you need more than a single server.
- Can you just add more servers?
    - What should you have done yesterday to make sure you can scale quickly today?



Support change

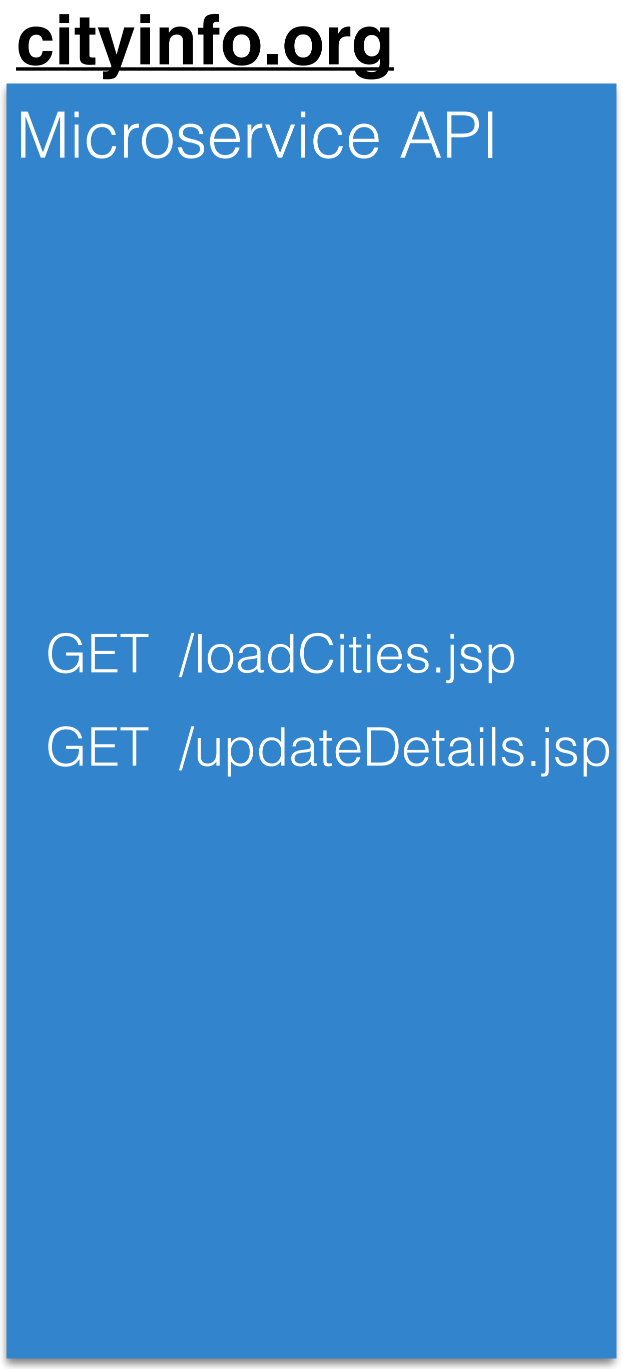
- Due to your popularity, your backend data provider just backed out of their contract and are now your competitor.
- The data you have is now in a different format.
- Also, you've decided to migrate your backend from PHP to node.js to enable better scaling.

- How do you update your backend without breaking all of your clients?



Support reuse

- You have your own frontend for cityinfo.org. But everyone now wants to build their own sites on top of your city analytics.
- Can they do that?



Design Considerations for Microservice APIs

- API: What requests should be supported?
- Identifiers: How are requests described?
- Errors: What happens when a request fails?
- Heterogeneity: What happens when different clients make different requests?
- Caching: How can server requests be reduced by caching responses?
- Versioning: What happens when the supported requests change?

REST: REpresentational State Transfer

- Defined by Roy Fielding in his 2000 Ph.D. dissertation
    - Used by Fielding to design HTTP 1.1 that generalizes URLs to URIs
    - http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
- "Throughout the HTTP standardization process, I was called on to defend the design choices of the Web. That is an extremely difficult thing to do… I had comments from well over 500 developers, many of whom were distinguished engineers with decades of experience. That process honed my model down to a core set of principles, properties, and constraints that are now called REST."
- Interfaces that follow REST principles are called RESTful

Properties of REST

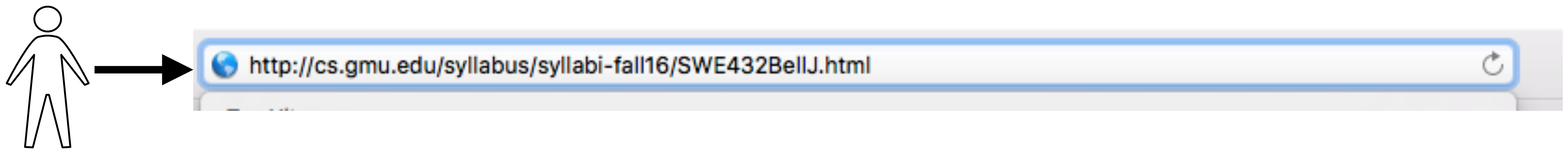
- Performance
- Scalability
- Simplicity of a Uniform Interface
- Modifiability of components (even at runtime)
- Visibility of communication between components by service agents
- Portability of components by moving program code with data
- Reliability

Principles of REST

- Client server: separation of concerns (reuse)
- Stateless: each client request contains all information necessary to service request (scaling)
- Cacheable: clients and intermediaries may cache responses. (scaling)
- Layered system: client cannot determine if it is connected to end server or intermediary along the way. (scaling)
- Uniform interface for resources: a single uniform interface (URIs) simplifies and decouples architecture (change & reuse)

HTTP: HyperText Transfer Protocol

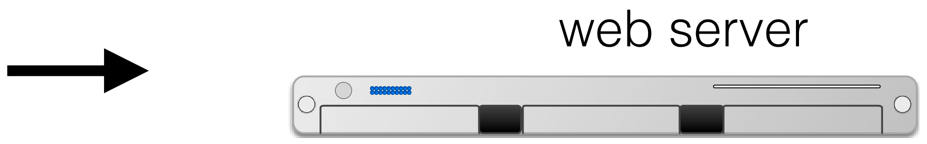
High-level protocol built on TCP/IP that defines how data is transferred on the web



http://cs.gmu.edu/syllabus/syllabi-fall16/SWE432BellJ.html

HTTP Request

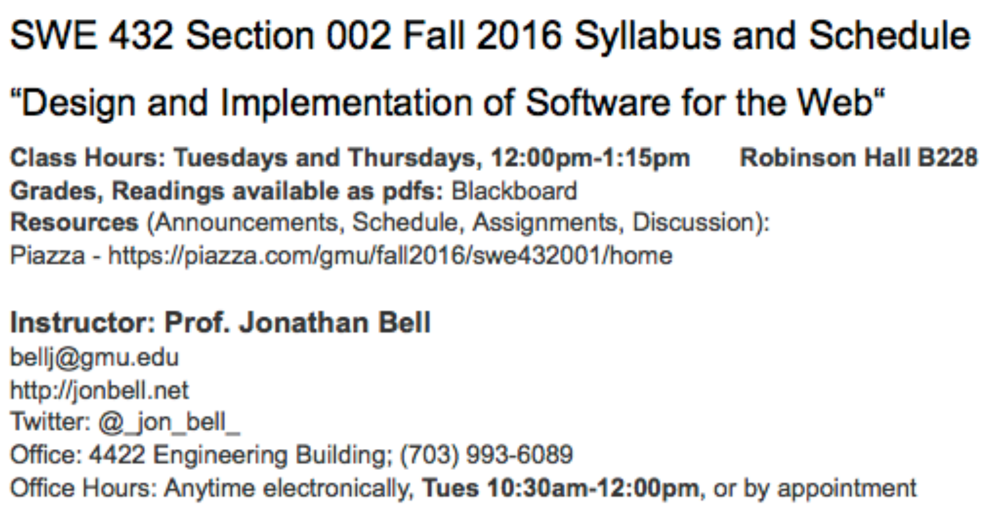
```
GET /syllabus/syllabi-fall16/SWE432BellJ.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```



Reads file from disk

HTTP Response

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

<html><head>...
```



SWE 432 Section 002 Fall 2016 Syllabus and Schedule

"Design and Implementation of Software for the Web"

Class Hours: Tuesdays and Thursdays, 12:00pm-1:15pm **Robinson Hall B228**
Grades, Readings available as pdfs: Blackboard
Resources (Announcements, Schedule, Assignments, Discussion):
Piazza - https://piazza.com/gmu/fall2016/swe432001/home

Instructor: Prof. Jonathan Bell
bellj@gmu.edu
http://jonbell.net
Twitter: @_jon_bell_
Office: 4422 Engineering Building; (703) 993-6089
Office Hours: Anytime electronically, **Tues 10:30am-12:00pm**, or by appointment

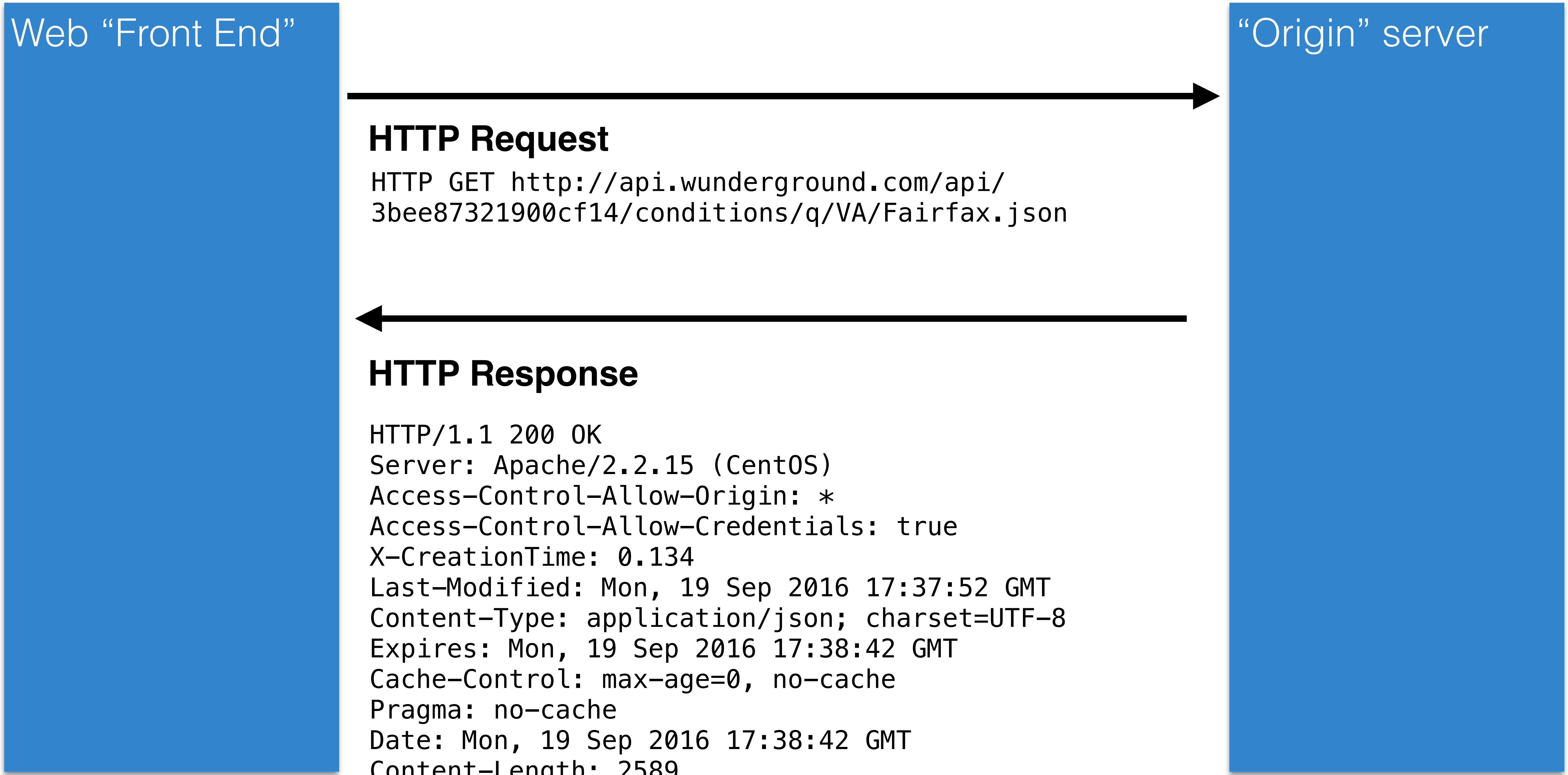
Uniform Interface for Resources

- Originally files on a web server
  - URL refers to directory path and file of a resource
- But… URIs might be used as an identity for any entity
  - A person, location, place, item, tweet, email, detail view, like
  - *Does not matter* if resource is a file, an entry in a database, retrieved from another server, or computed by the server on demand
  - Resources offer an *interface* to the server describing the resources with which clients can interact

URI: Universal Resource Identifier

- Uniquely describes a resource
  - https://mail.google.com/mail/u/0/#inbox/157d5fb795159ac0
  - https://www.amazon.com/gp/yourstore/home/ref=nav_cs_ys
  - http://gotocon.com/dl/goto-amsterdam-2014/slides/StefanTilkov_RESTIDontThinkItMeansWhatYouThinkItDoes.pdf
  - Which is a file, external web service request, or stored in a database?
    - It does not matter
- As client, only matters what actions we can *do* with resource, not how resource is represented on server

Intermediaries



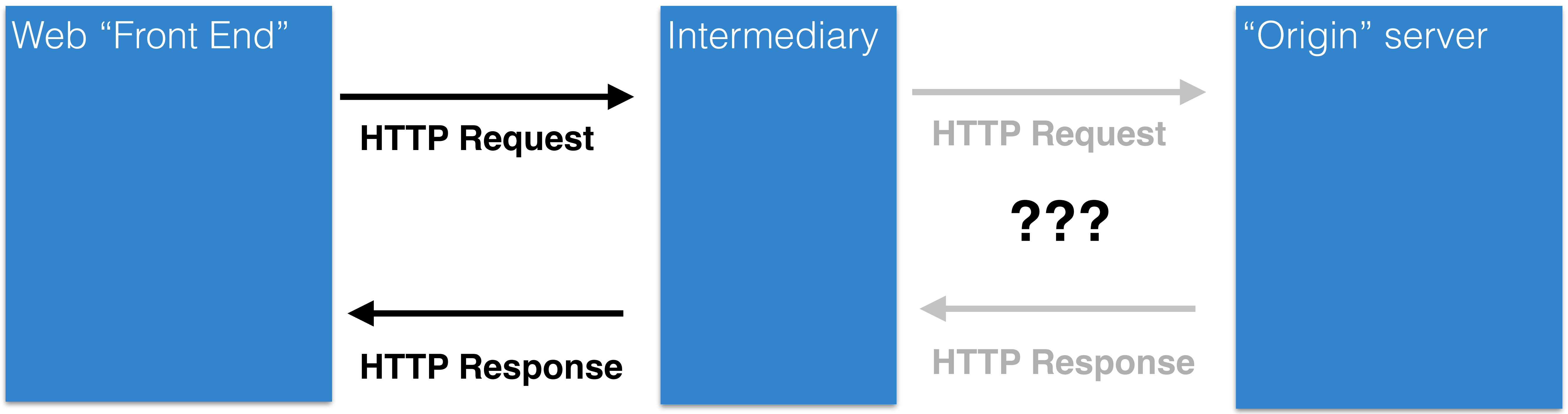
HTTP Request

```
HTTP GET http://api.wunderground.com/api/
3bee87321900cf14/conditions/q/VA/Fairfax.json
```

HTTP Response

```
HTTP/1.1 200 OK
Server: Apache/2.2.15 (CentOS)
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
X-CreationTime: 0.134
Last-Modified: Mon, 19 Sep 2016 17:37:52 GMT
Content-Type: application/json; charset=UTF-8
Expires: Mon, 19 Sep 2016 17:38:42 GMT
Cache-Control: max-age=0, no-cache
Pragma: no-cache
Date: Mon, 19 Sep 2016 17:38:42 GMT
Content-Length: 2589
Connection: keep-alive


{
  "response": {
  "version":"0.1",
  "termsofService":"http://www.wunderground.com/weather/api/d/terms.html",
```

Intermediaries



- Client interacts with a resource identified by a URI
- But it never knows (or cares) whether it interacts with origin server or an unknown intermediary server
  - Might be randomly load balanced to one of many servers
  - Might be cache, so that large file can be stored locally
    - (e.g., GMU caching an OSX update)
  - Might be server checking security and rejecting requests

Challenges with intermediaries

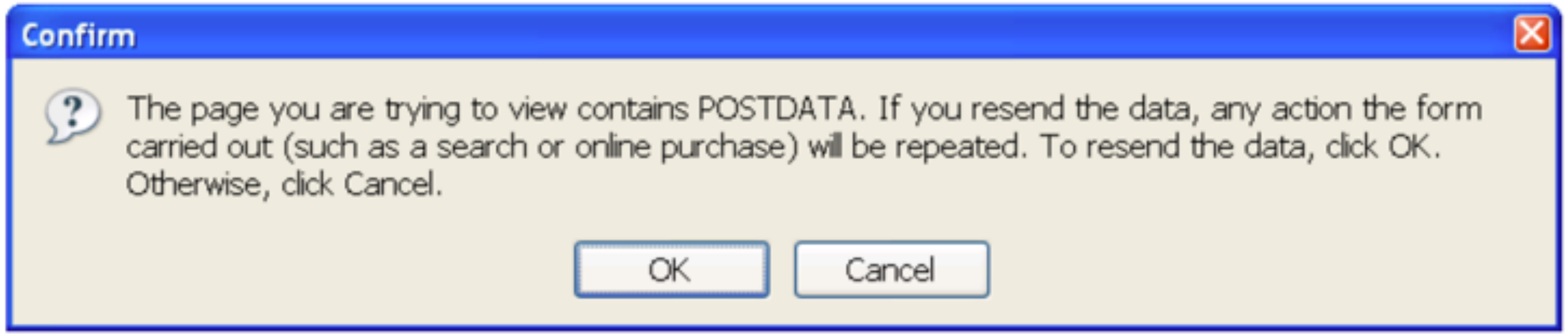
- But can all requests really be intercepted in the same way?
  - Some requests might produce a change to a resource
    - Can’t just cache a response… would not get updated!
  - Some requests might create a change every time they execute
    - Must be careful retrying failed requests or could create extra copies of resources

HTTP Actions

- How do intermediaries know what they can and cannot do with a request?
- Solution: HTTP Actions
  - Describes what will be done with resource
  - GET: retrieve the current state of the resource
  - PUT: modify the state of a resource
  - DELETE: clear a resource
  - POST: initialize the state of a new resource

HTTP Actions

- GET: safe method with no side effects
  - Requests can be intercepted and replaced with cache response
- PUT, DELETE: idempotent method that can be repeated with same result
  - Requests that fail can be retried indefinitely till they succeed
- POST: creates new element
  - Retrying a failed request might create duplicate copies of new resource


Confirm

The page you are trying to view contains POSTDATA. If you resend the data, any action the form carried out (such as a search or online purchase) will be repeated. To resend the data, click OK. Otherwise, click Cancel.