Introduction to REST APIs

Definitions

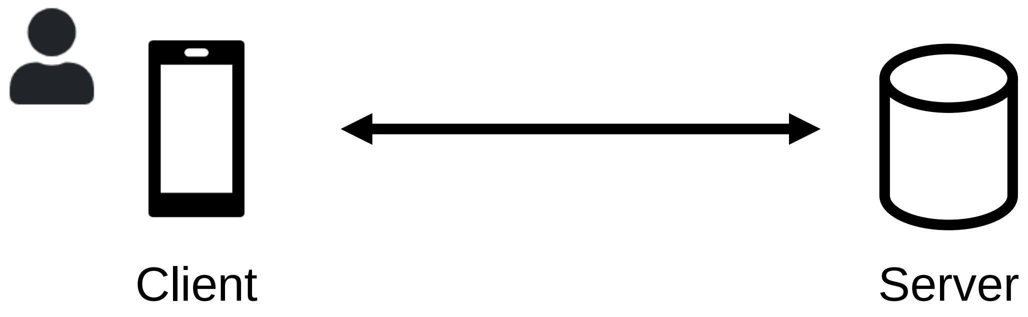
What is a Web API?
API stands for Application Programming Interface. A web API allows specifically exposed methods of an application to be accessed and manipulated outside of the program itself. Web APIs use web protocols (HTTP, HTTPS, JSON, XML, etc.). For example, a web API can be used to obtain data from a resource (such as U.S. postal service zip codes) without having to actually visit the application itself (checking usps.com).

What is REST?
REST is short for Representational State Transfer. REST is a software architectural style… a set of rules and conventions for the creation of an API.

A computer scientist by the name of Roy Fielding defined the principles of REST in his 2000 PhD dissertation. Like any theoretical ideal, there are a lot of practical exceptions to Fielding's principles. But the principles of REST are intended to make APIs and systems that are: efficient, scalable, simple, reliable and modifiable (i.e. future-proof) among other things … all excellent ideals!

The 6 constraints of REST (1)

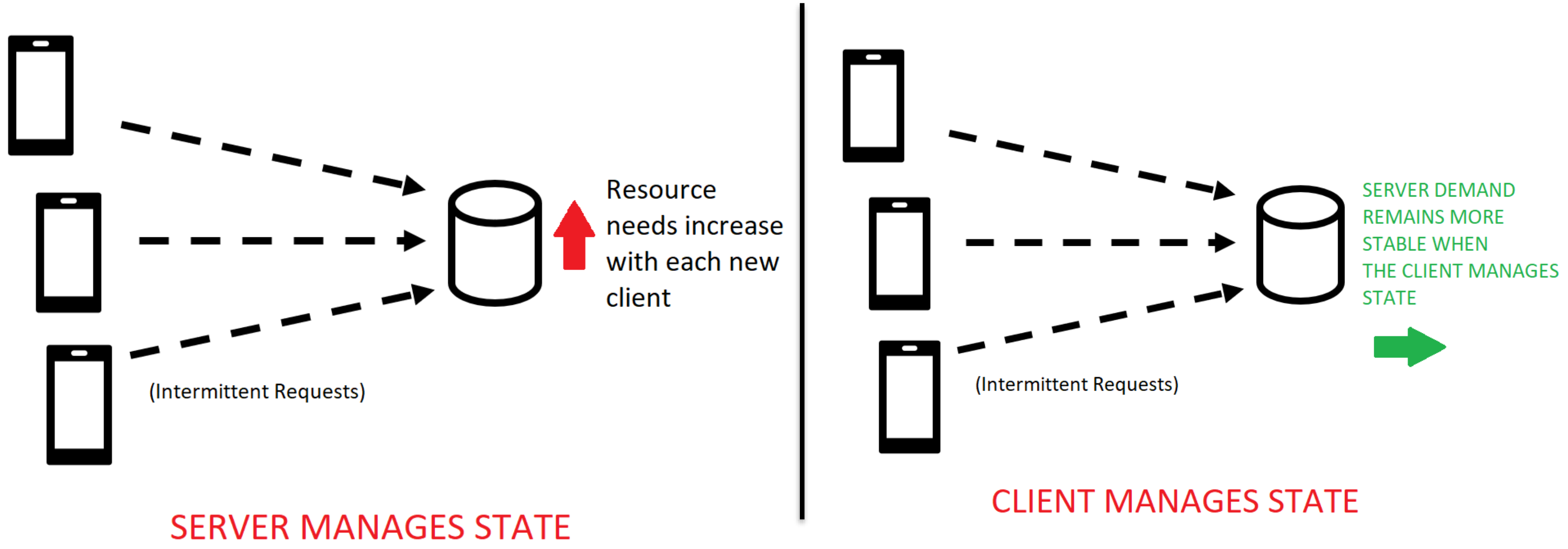
1. **Client-Server architecture –** Simply put, RESTful systems separate the systems responsible for storing and processing the data (the server) from the systems responsible for collecting, requesting, consuming, and presenting the data to a user (the client). This separation should be so distinct that the client and server systems can be improved and updated independently each other.



An upgrade to the client should not necessitate an upgrade to the server… and vice versa.

The 6 constraints of REST (2)

2. **Statelessness –** As far as the server is concerned, all client requests are treated equally. There's no special, server-side memory of past client activity. The responsibility of managing state (for example, logged in or not) is on the client. This constraint is what makes the RESTful approach so scalable.

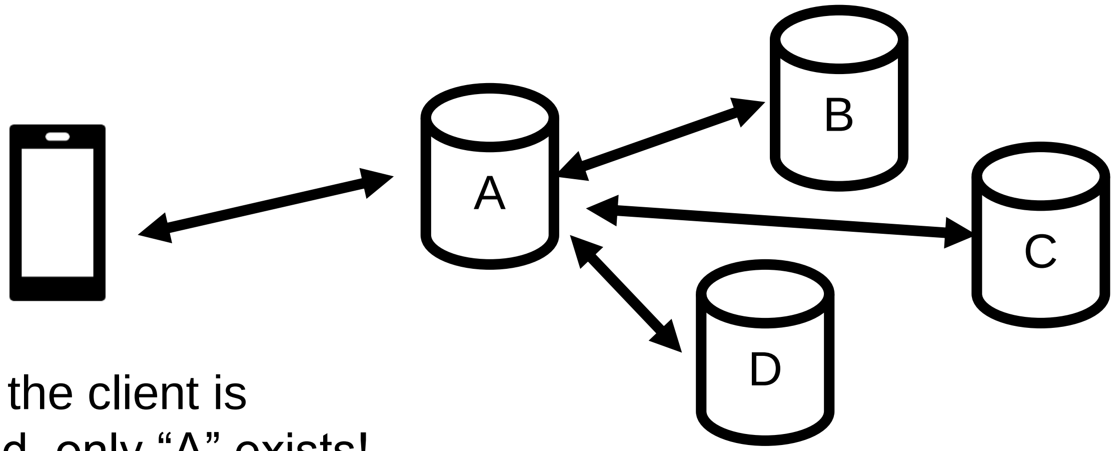

Resource needs increase with each new client
(Intermittent Requests)
SERVER MANAGES STATE
SERVER DEMAND REMAINS MORE STABLE WHEN THE CLIENT MANAGES STATE
(Intermittent Requests)
CLIENT MANAGES STATE

Be advised! In REST each and every resource request is to convey the application state. That means the state gets transferred with each request!

The 6 constraints of REST (3)

3. **Cacheability –** Clients and servers should be able to cache resource data that changes infrequently. For example, there are 52 states and other jurisdictions in the U.S.A. That's not likely to change soon. So, it is inefficient to build a system that queries a database of states each and every time you need that data. Clients should be able to cache that infrequently updated date and web servers should be able to control the duration of that cache.

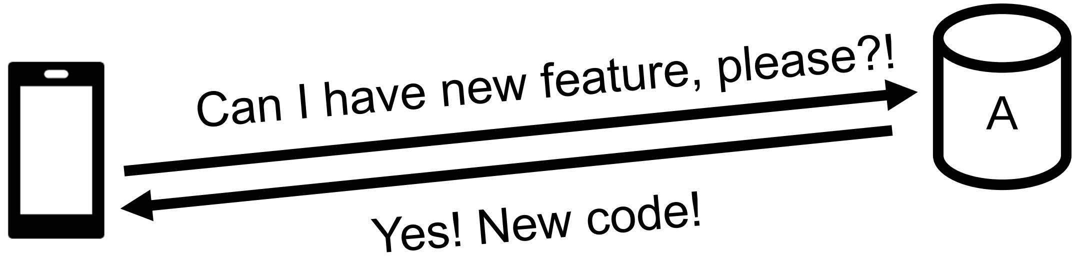
4. **Layered system –** A client cannot tell whether it is connected directly to an end server, or to an intermediary along the way. Intermediary servers can also improve system scalability.



As far as the client is concerned, only "A" exists!

The 6 constraints of REST (4)

5. **Code on demand (Optional) –** Servers can temporarily extend or customize the functionality of a client by transferring executable code. This is constraint is optional.



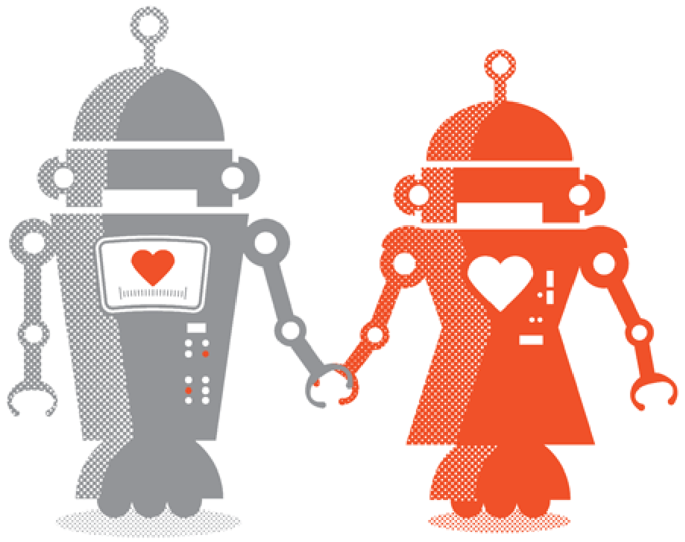
6. **Uniform Interface –** This constraint is comprised of 4 additional “constraints” or “principles”. Taken together, these 4 additional principles basically require the API to be consistent with HTTP standards and self describing. That means that, in theory, given an API’s web address, a server should respond with all the available actions and resources at that address. Each subsequent server response should then contain enough information to take additional actions on the resource. This constraint is very rarely realized in practice!

The marriage of REST and HTTP

Roy Fielding was also one of the principle authors of the HTTP specification. Fielding created the REST constraints with HTTP in mind.

Part of the power of the REST architectural style is that HTTP was already widely in use before the REST constraints were formally defined.

In theory, the six REST constraints we just reviewed could be applied to other protocols, but in practice RESTful APIs are HTTP based.



RESTful API conventions – HTTP Methods

* GET – The get method is used to retrieve data from a resource. According to RESTful conventions, GETs are safe to execute over and over. For example: It is safe to run the following GET request as many times as you want:

https://www.amazon.com/s/ref=nb_sb_noss_1?url=search-alias%3Daps&field-keywords=robot

* POST – The POST method would be used to create a data record, or initiate an action. Imagine each POST as being a request to make a amazon purchase. You would want to be careful about doing that more times than necessary!

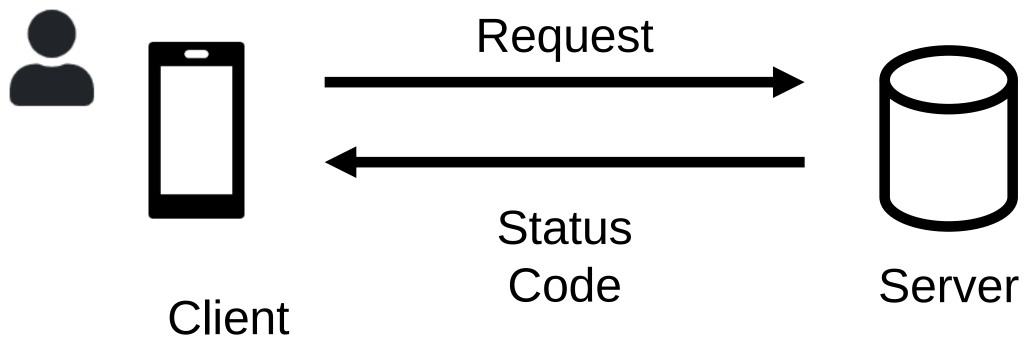
PUT -- The PUT method exists in HTTP. It should be used to update an existing data record.

DELETE – The DELETE method exists in HTTP. It should be used to delete a data record.

* These are the only two methods you will use this semester.

RESTful API conventions – HTTP Status Codes

Every HTTP request results in a status code to be sent back to the client.



2xx Status codes indicate success. The most common status code is 200, OK.

3xx Status codes indicate that the client needs to do something else to complete the request. Usually this means making the request from a different location. Status code 301 means the resource has been permanently moved somewhere else.

HTTP Status Codes continued

4xx Status codes indicate a client error. That is, the client sent a request that did not make sense.

A classic example of this is “404 Not Found”.   Try it! Go to https://www.google.com/bad then try going to https://www.temple.edu/bad Notice that both pages report the code 404. That’s not just a coincidence.

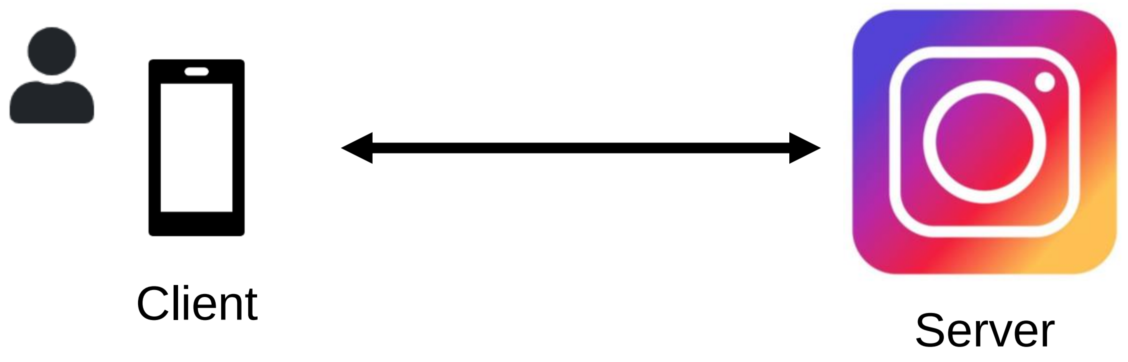
The internet is built on standards like HTTP and these error codes are part of that standard.

5xx Status codes indicate a server error. That means that the request made by the client appears to be fine. But the server is experiencing some difficulty.

A server experiencing some difficulty might report “500 Internal server error”

More thorough documentation of the HTTP status codes can be found here: https://www.restapitutorial.com/httpstatuscodes.html

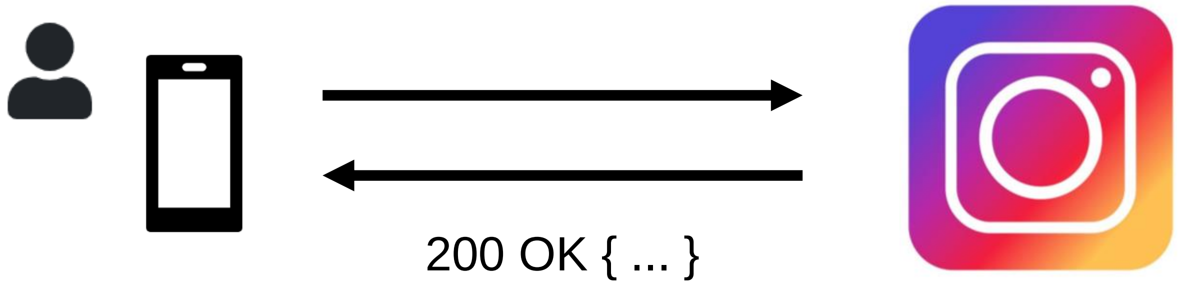
An Example



An Example

GET

https://www.instagram.com/**profile?user=somemadeupuser**



200 OK { ... }

Client

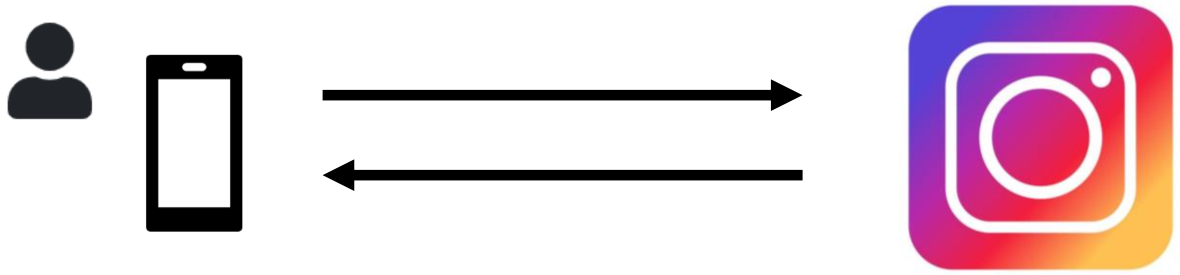
Server

OR

404 Not Found

An Example

POST
https://www.instagram.com/**comment**
{ picture_id: <id>, comment: "💦" }



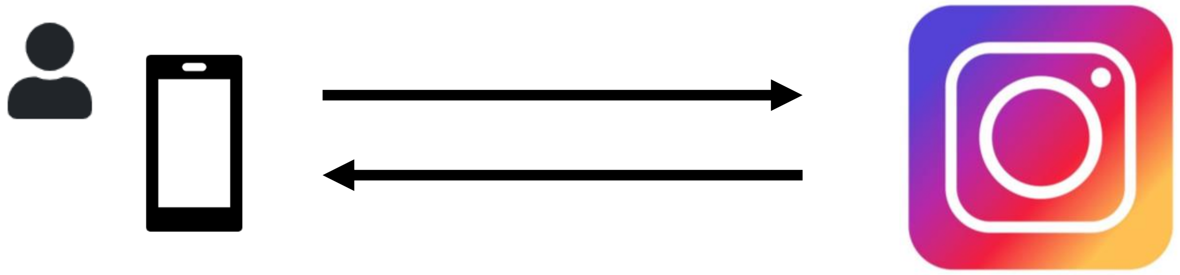
Client

200 OK
{ comment_id: <id>,
commenter_id: <id>,
picture_id: <id>,
comment: "💦" }

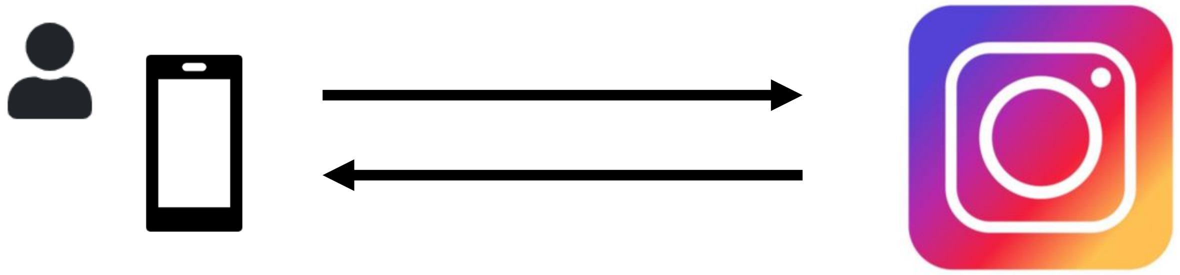
Server

An Example

POST
https://www.instagram.com/**comment**
{ picture_id: 0, comment: "💦" }



An Example



POST
https://www.instagram.com/**comment**
{ picture_id: <id>, comment: "" }

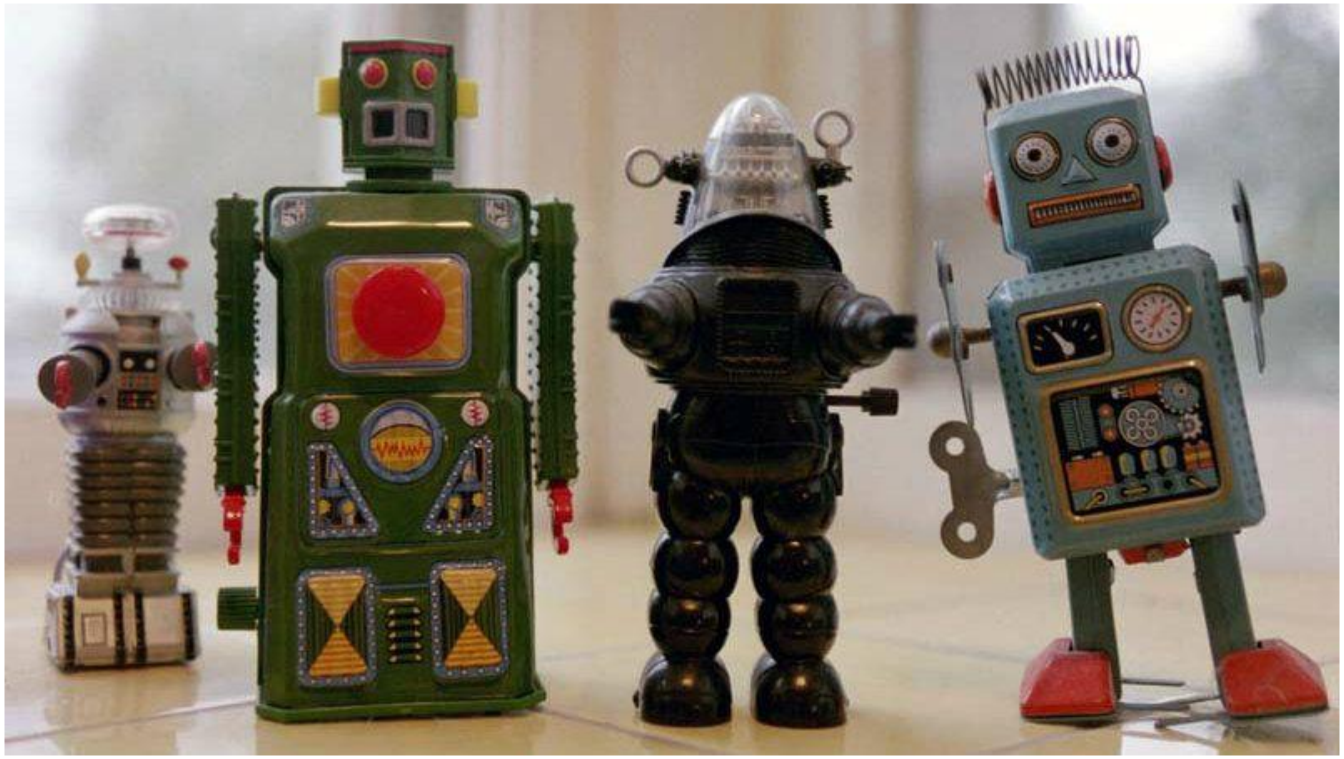
Client

400 Bad Request

Server

Comment text must not be empty

A quick reality check…



The REST design constraints are good in theory. However, ***in practice*** no one … not even the largest technology companies … implements them perfectly.   Three of the most common constraint violations follow.

Wrinkles and exceptions

1. Abuse of the GET method.

   Again, GET is assumed to be a safe operation that never modifies the state of the resource. However, it has been often contorted into performing other functions… for example:

   http://exampleapi.xyz?action=ADD&productname=Widget

2. APIs are not self-documenting

   This RESTful ideal has yet to be fully realized. Even so, some API providers don't even try!

3. State matters – In the REST design, the burden of managing state is on the client. That, however, raises many security concerns. API providers like the efficiency gains made possible by REST … but there is no one consistent approach to managing state.
   Next class, we will explore ***one*** of those approaches.

Thanks for your attention!