CS 4220

- Current Trends in Web Design & Development -

Prof. Cydney Auman

AGENDA

01 HTTP Verbs, Params & Codes

02 REST and RESTful

03 CORs

04 REST Client & Code Demo

HTTP Verbs

There are four basic **HTTP verbs** used in making requests to interact with Servers.

- **GET** — Retrieving data. This can be used to retrieve a list of data with optional filtering. Or by using an id to get a specific piece of data.
- **POST** — Creating a new data entry by sending the details in the POST body.
- **PUT** — Updating a specific data entry by using an id and sending the details in the PUT body.
- **DELETE** — Removing a specific data entry by using an id. Typically the id is passed in the url parameters.

HTTP Path and Query Parameters

Path Parameters

Path Parameters are used to specify a particular resource and/or resources often times by using an id. They are always part of the full URL path.

Example:

```
https://deckofcardsapi.com/api/deck/:deck_id/draw
```

Query Parameters

Query Parameters are typically used to filter, sort, apply a flag and etc when making requests. They are always at the end of the URL to the right of the ? and each query parameter is set equal to its value. Multiple query parameters are separated by an &.

Example:

```
https://deckofcardsapi.com/api/deck/new/?deck_count=1&jokers_enabled=true
```

HTTP Response Codes

Common Status Codes and Meanings
https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

200 - OK
The request has succeeded.

201 - CREATED
The request has succeeded and a new resource has been created as a result.

400 - BAD REQUEST
The server could not understand the request due to some invalid syntax.

404 - NOT FOUND
The server can not find the requested resource. Or the URL is not recognized.

500 - INTERNAL SERVER ERROR
The server encountered a situation it doesn't know how to handle. Likely an error or bug in the server code

REST and RESTful

REST meaning **RE**presentational **S**tate **T**ransfer is an architectural style for servers that is guided by a set of standard principles. Services that implement this architecture and abide by these principles are considered to be **RESTful** services.

- Client-Server Separation
- Stateless
- Cache Constraints
- Uniform Interface

Client-Server Separation

In the **REST** architecture the implementation of the client and the server are independent of each other.

This means that things that are client side concerns suchs a user interface and design can be changed without impacting the server.   And this also means that things that are server side concerns such as data storage and hosting can be changed without impacting the client.

This separation creates a dividing line that allows both the client side and the server side to evolve and scale independently.

This *does not mean* that the server can abruptly change the API meaning it cannot just change agreed upon interface for how the client interacts with the server.

Stateless

In the **REST** architecture the server side is stateless.   This means that the server does not maintain or track state of the client. The server never needs to know about the state of the client.

Every request from the client to server must contain all of the information necessary for the server to understand and process the request. And in-turn the server will provide a response always containing the necessary information to process requests.

Because of this guiding principle the client side must store any needed session states or token and use them at the time a request is made.

Cache Constraints

In the **REST** architecture there is a Cache Constraint.

Interestingly, in the way that this constraint is defined is that is *does not* mandate or even require the server to provide a cached response of data.

It simply means that the server must label all responses as either cacheable or non-cacheable. If a response is indicated as cacheable then the client can choose to reuse that data response. Because caching often times leads to performance improvements the client side will indeed make use of cacheable data.

Uniform Interface

In the **REST** architecture there must be Uniform Interface and this principle is seen as one of the key aspects of the REST architecture. Every interaction (request/response) must communicate in the same way. Meaning...

First each resource that a server provides must have its own endpoint (URL). And these endpoints should represent the various actions of GET, POST, PUT, and DELETE per resource. Each method then operates on one and only one resource.

Each response includes information that indicates to the client how to process or parse the data. For instance often time responses are indicated as `application/json`.

Furthermore, when the client gets a data response from a resource endpoint it must include enough information that allows the client to update and/or delete the resource on the server (if the client has such permissions).

CORS

CORS or Cross-Origin Resource Sharing is a mechanism that uses additional HTTP headers to inform the browser to allow a web application running on one origin to have permission to access resources from a server located at a different origin.

With CORS an additional header is included from the server which is a way of letting the client know that it can use the response data that it received.

One key header used for CORS is `Access-Control-Allow-Origin: <origin>`
Where `<origin>` represent the origin (server location) of the client's request. In the case where a server will allow all clients to access resources and make use of data the origin is specified as **`*`**.

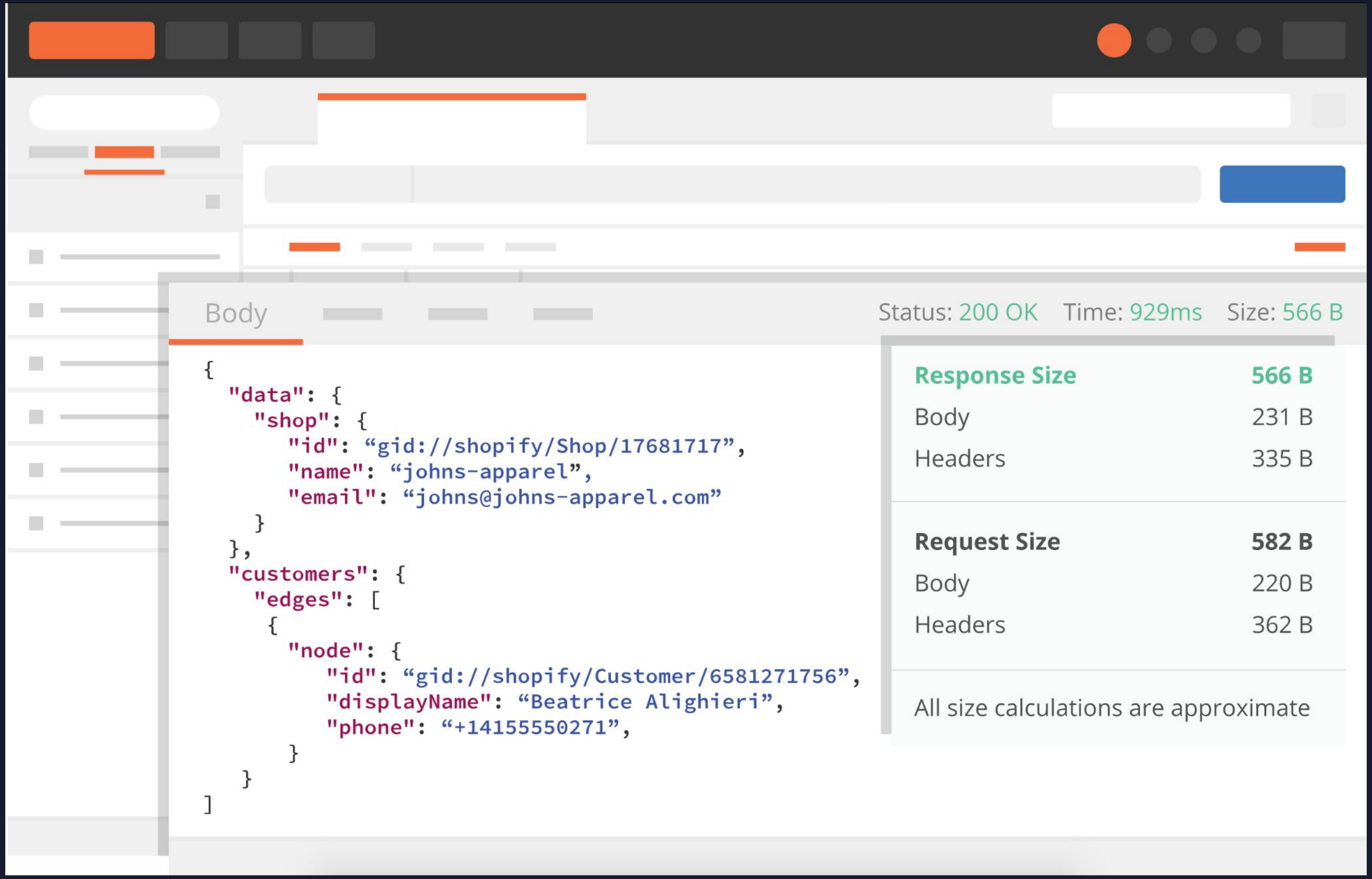
REST Client - POSTMAN

A REST Client is a development tool that allows us to easily create and send API calls to a server. REST Clients should support all HTTP method types and allow you to customize headers

After the request is sent REST Clients allow us to view the status code, response time, and overall response size.

Other features included are automatic language detection, syntax highlighting, search, and text formatting which makes it easier to view the response body.

https://www.postman.com/product/rest-client/



Body

Status: 200 OK Time: 929ms Size: 566 B

```
{
  "data": {
    "shop": {
      "id": “gid://shopify/Shop/17681717”,
      "name": “johns-apparel”,
      "email": “johns@johns-apparel.com”
    }
  },
  "customers": {
    "edges": [
     {
      "node": {
        "id": “gid://shopify/Customer/6581271756”,
        "displayName": “Beatrice Alighieri”,
        "phone": “+14155550271”,
      }
    }
]
```

Response Size	**566 B**
Body	231 B
Headers	335 B
Request Size	**582 B**
Body	220 B
Headers	362 B

All size calculations are approximate

```
console.log(‘Week 12’);
console.log(‘Code Examples’);
```

Review and Prep

Review

- Review Slides & Run Code Examples
- Read More on HTTP Methods
  https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods
- Read More on HTTP Status Codes
  https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Preparation for Next Class

- Quiz on Thursday
- Read about Express Middleware
  https://expressjs.com/en/guide/using-middleware.html
- Download a REST Client
  https://www.postman.com/product/rest-client/