Networking and HTTP
Consuming RESTful APIs

Module 2 - Week 6

Schedule

Week	Topics
Week 0	Module 2 Orientation / PostgreSQL
Week 1	Intro to databases / Ordering, limiting, and grouping
Week 2	SQL joins / Insert, update, and delete / Database design
Week 3	Data Access / Data security
Week 4	DAO testing
Week 5	**Mid-module project**
Week 6	Postman / NPM / Networking and HTTP / Consuming RESTful APIs
Week 7	Server-side APIs
Week 8	Securing APIs
Week 9	**End-of-module project**
Week 10	**Assessment**

Primary objectives

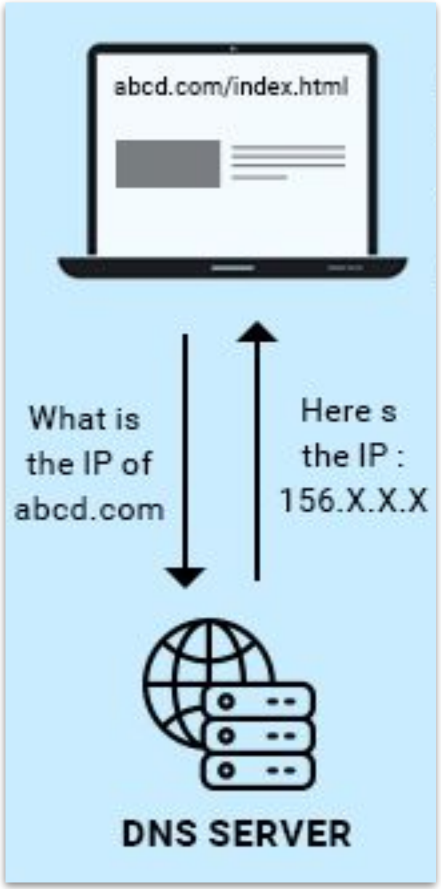
- Explain the purpose of a DNS and IP Addresses.
- Identify and explain parts of a URL.
- Identify and explain the purpose of the main components of HTTP Requests and Responses.
- Explain the steps of a typical HTTP request between a web browser and a server.
- Explain what JSON is and how it is used in a Java / C# program.
- Identify resources in a RESTful API, describe the relationship between a resource and a URL.
- Explain how the different HTTP methods are used to interact with RESTful resources.
- Use Postman to make a GET, PUT, POST, or DELETE request and inspect the result
- Make HTTP GET, POST, PUT, an DELETE requests to a RESTful web service using Java and process the response.
- Handle an error from RESTful API requests

DNS (Domain Name System)

IP Addresses are understood by the computer, but are not easy to use for humans.

The **Domain Name System (DNS)** service allows for easy to remember names for humans to use. It works like a phone book, we give it the human relatable name and it returns the IP address for the computer to use.

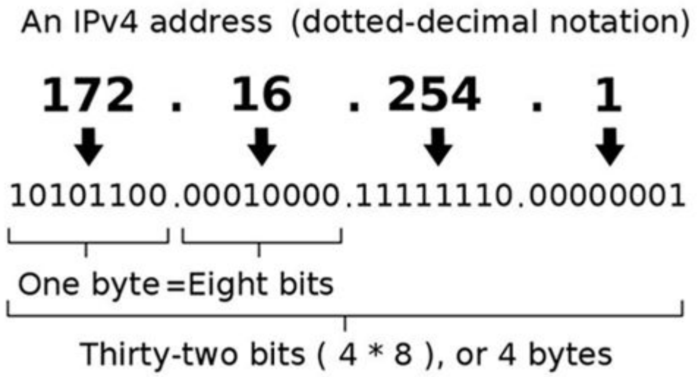
Domain Name: techelevator.com
IP Address: 198.49.23.144



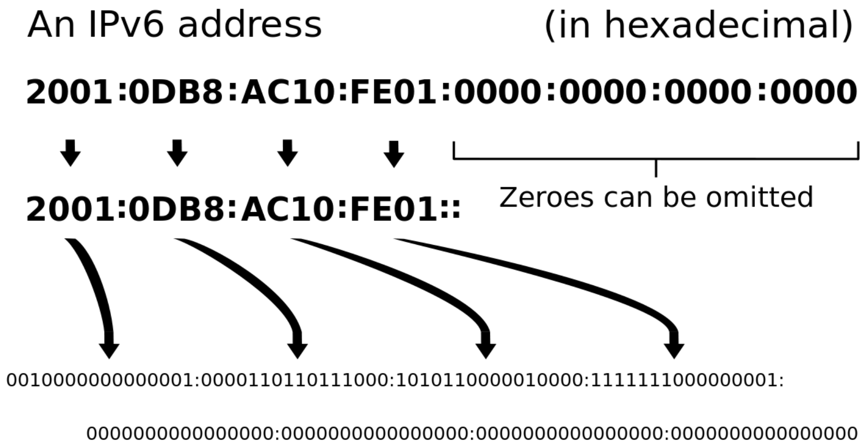
IP (Internet Protocol) Addresses

An **IP Address** identifies a computer or device on a network (including the internet).

All devices/computers on a network must have an IP address to communicate with other devices/computers on the network. It gives the location of the device like the street address of a house.



An IPv4 address (dotted-decimal notation)
172 . 16 . 254 . 1
10101100.00010000.11111110.00000001
One byte=Eight bits
Thirty-two bits (4 * 8), or 4 bytes



IPv6 addresses are 128 bits long, written as 8 sections of 16 bits each.

URL (Universal Resource Locator)

A URL (Universal Resource Locator) tells a client how to make a request to a server for a specific resource.

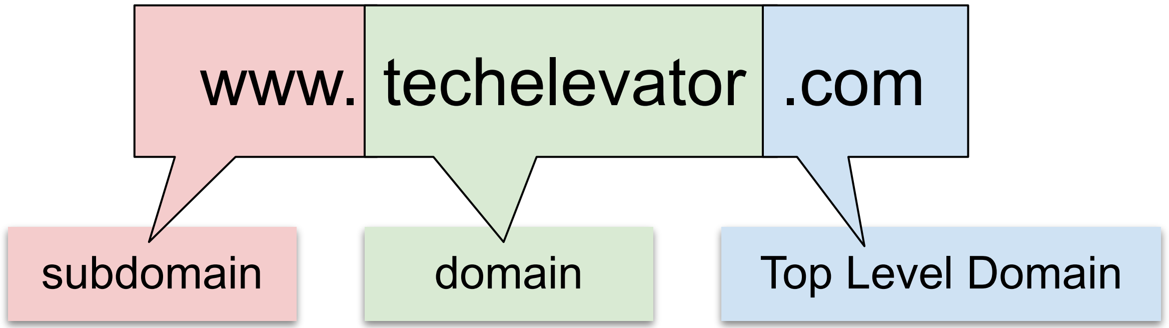
http://www.techelevator.com:80/events/current?month=march&day=27#00h02m30s

http://	**Protocol** (the language being spoken between client and server)
www.techelevator.com	**Domain** (the address of the computer the server is on)
:80	**Port** (identifies the server on the hosting computer)
/events/current	**Path to resource** (what is being requested)
?	**Query** (indicates parameters are being sent)
month=march&day=27	**Parameters** (keys with values for the server to use)
#00h02m30s	**Anchor/Fragment** (information that the server should return to the client)

Domain Names

Domain names are composed of parts, each separated by a period. Domains start a top level domain and then form a hierarchy of subdomains, each a child of the higher level domain.

Top Level Domains: .com, .net, .org, .gov, .io, …



Port

There can be multiple software **servers** running on a single computer. Port numbers are used to identify which server application should handle the request.

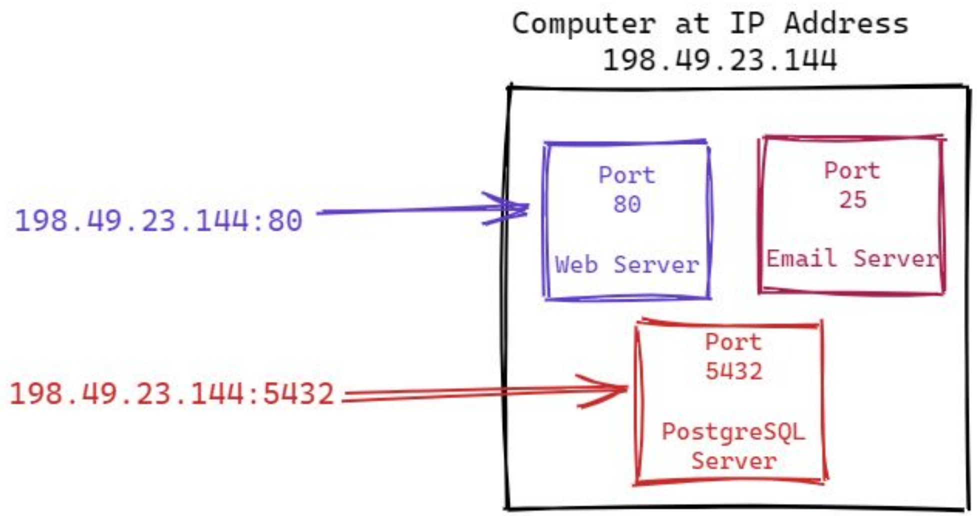
> *IP Address is like the street address of a building, and the port number would be equivalent to an apartment number in that building.*

The Port is added after the IP Address separated by a colon: 198.49.23.144:56

Port numbers range from 0-65535. Ports in the 0-1023 range are referred to as *well-known ports* and designated for specific uses.

Common well-known ports:

- Port 80 - HTTP
- Port 443 - HTTPS
- Port 25 - SMTP (Email Mail)
- Port 21 - FTP (File Transfer)
- Port 22 - SSH (Secure remote terminal)



HTTP (HyperText Transfer Protocol)

A **Protocol** defines the rules governing how clients and servers will communicate. It is a defined language and process that defines how a client should make a request and the server will return the response.

HTTP is the main **Application Protocol** used for the World Wide Web and is how browsers and web servers communicate.

HTTP communicates using a *stateless* **request and response**.

Parts of a HTTP Request

1. Method
2. Requested Resource
3. Header
4. Parameters

Parts of a HTTP Response

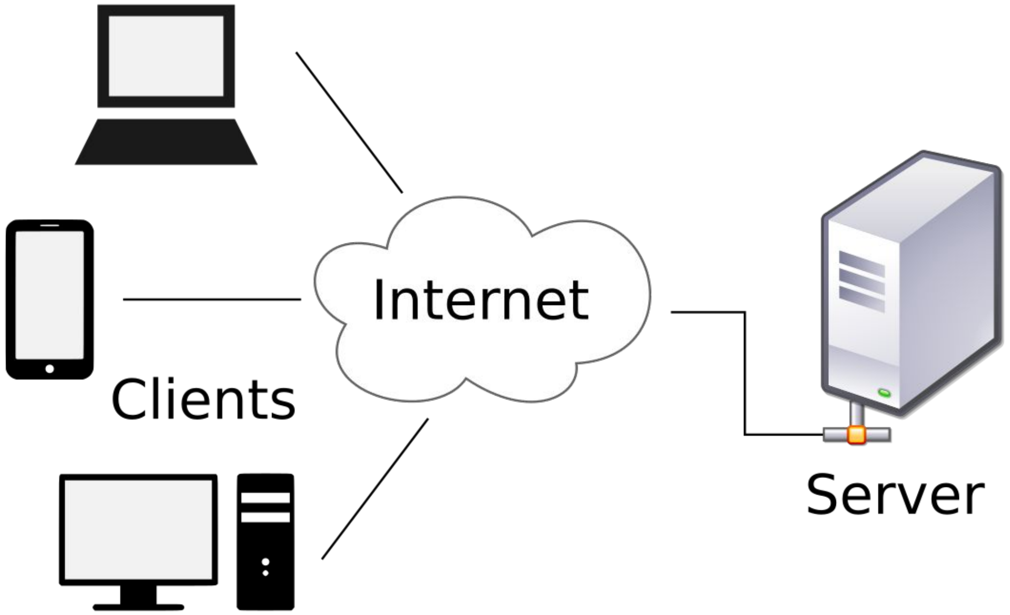
1. Status Code
2. Header
3. Content Type
4. Content (Body)

Clients and Servers

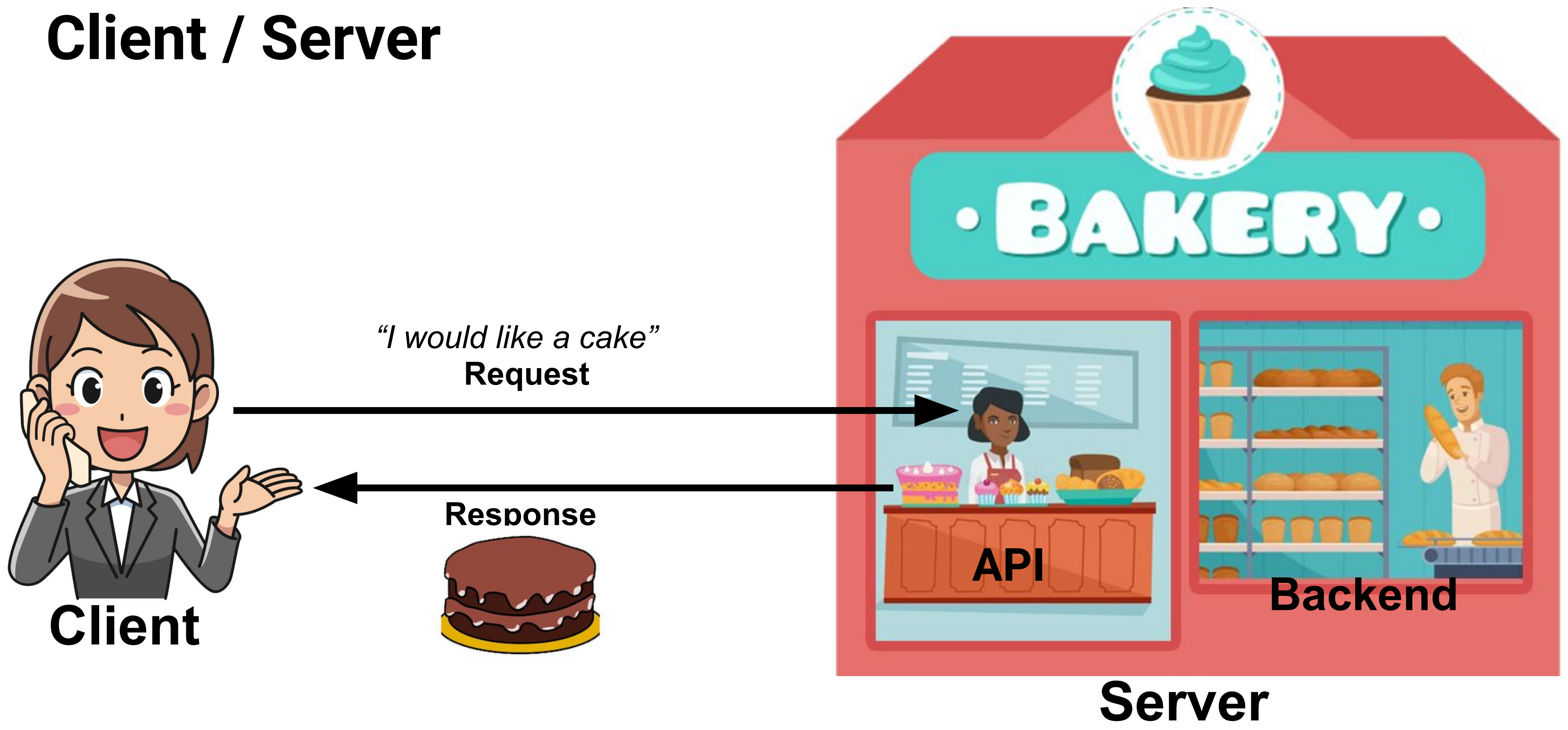
Server is a generic term referring to either software or hardware that processes request from **clients**. A server offers shared **resources** that can be requested for use by a **client**.

A **client** is software that sends a request to a server to access a shared resource and processes the response returned from the server.

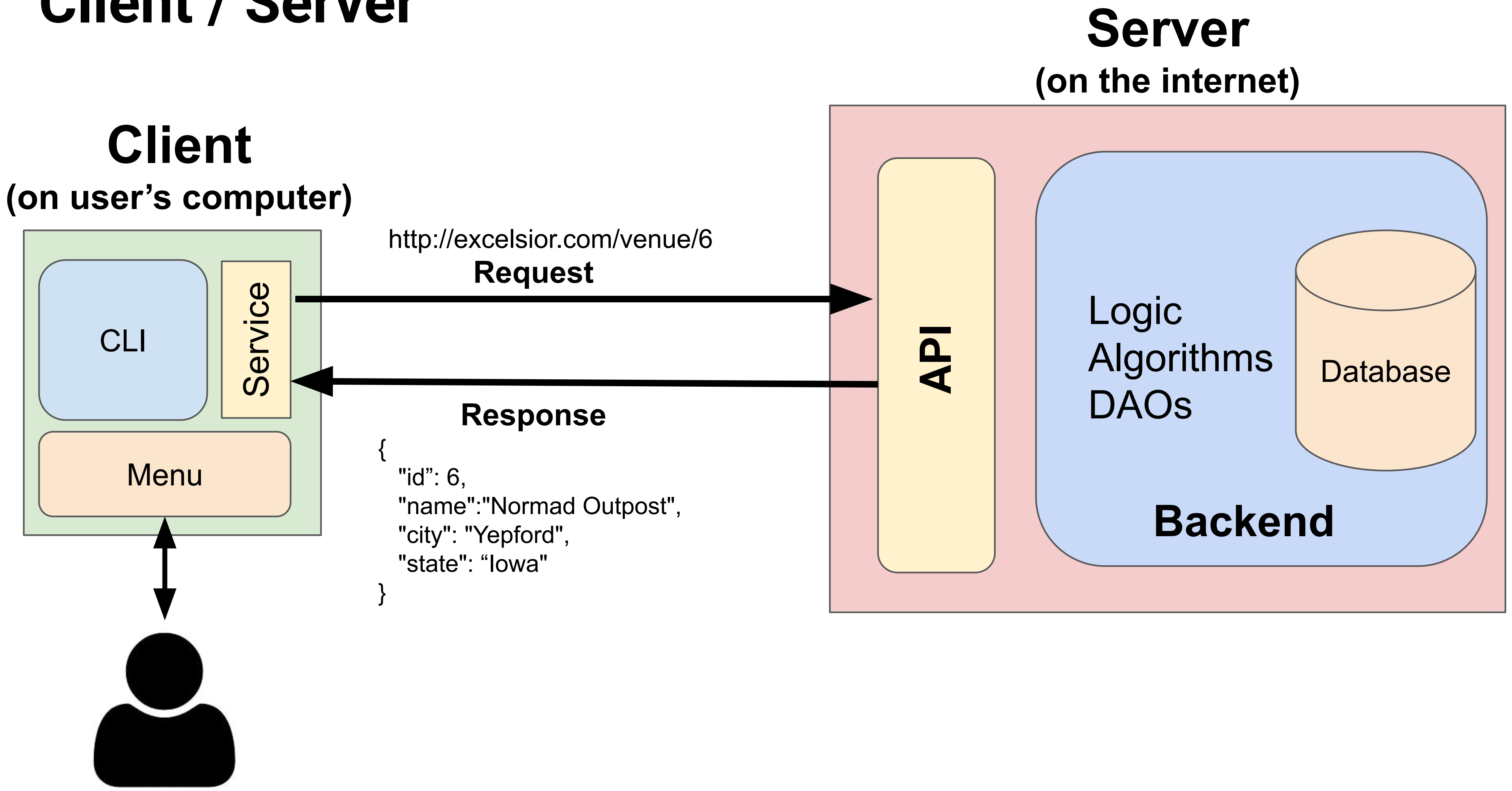
Client	**Server**
Web Browser	Web Server
Smart TV	Netflix's computers
Phone	Messaging Server
Email App	Email Server
PGAdmin 4	PostgreSQL



Client / Server



Client / Server



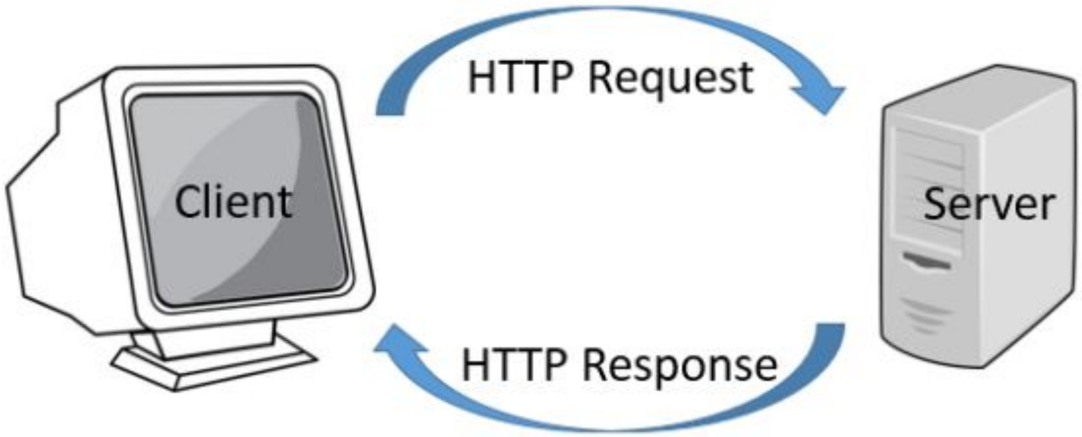
```
{
   "id": 6,
   "name":"Normad Outpost",
   "city": "Yepford",
   "state": "Iowa"
}
```

HTTP

- A Protocol defines the rules governing how clients and servers will communicate. It is a defined language and process that defines how a client should make a request and the server will return the response.
- HTTP (HyperText Transfer Protocol) is the main Application Protocol used for the World Wide Web and is how browsers and web servers communicate.
- HTTP communicates using a stateless request and response.
- Provides ability to implement authentication
- Resources access via Universal Resource Locators (URLs)
- Defines methods to indicate the desired action to be performed on the identified resource.
    - We will use:
        - GET
        - POST
        - PUT
        - DELETE

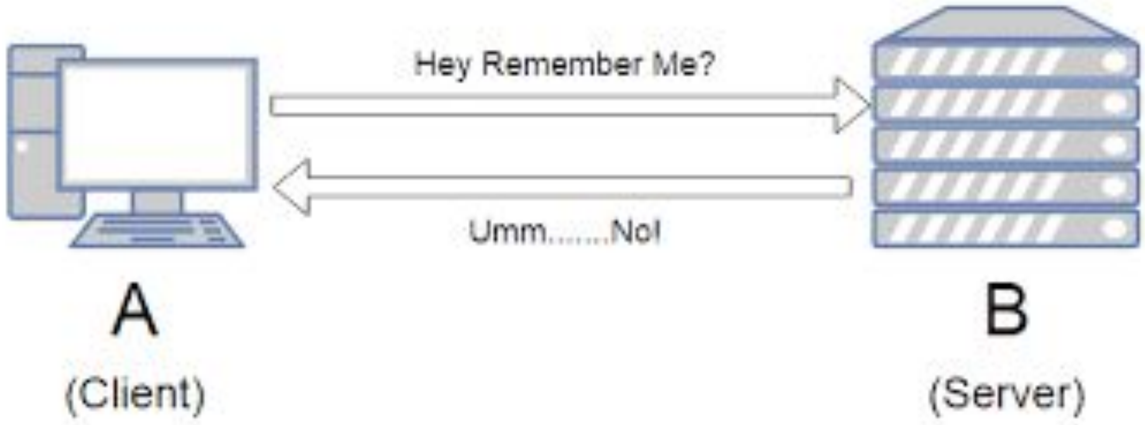
HTTP

- A client sends a request and the server replies with a response.
- The HTTP request/response pair is stateless. Meaning that each request/response is independent and without context of previous request/response traded between the same client and server.



Stateless

- Once a response is sent, the server does not retain any information about the request.
- The server does not remember that the client sent a previous request, any details of that request, or anything about the client. Each request is independent and must include all information needed by the server to respond.



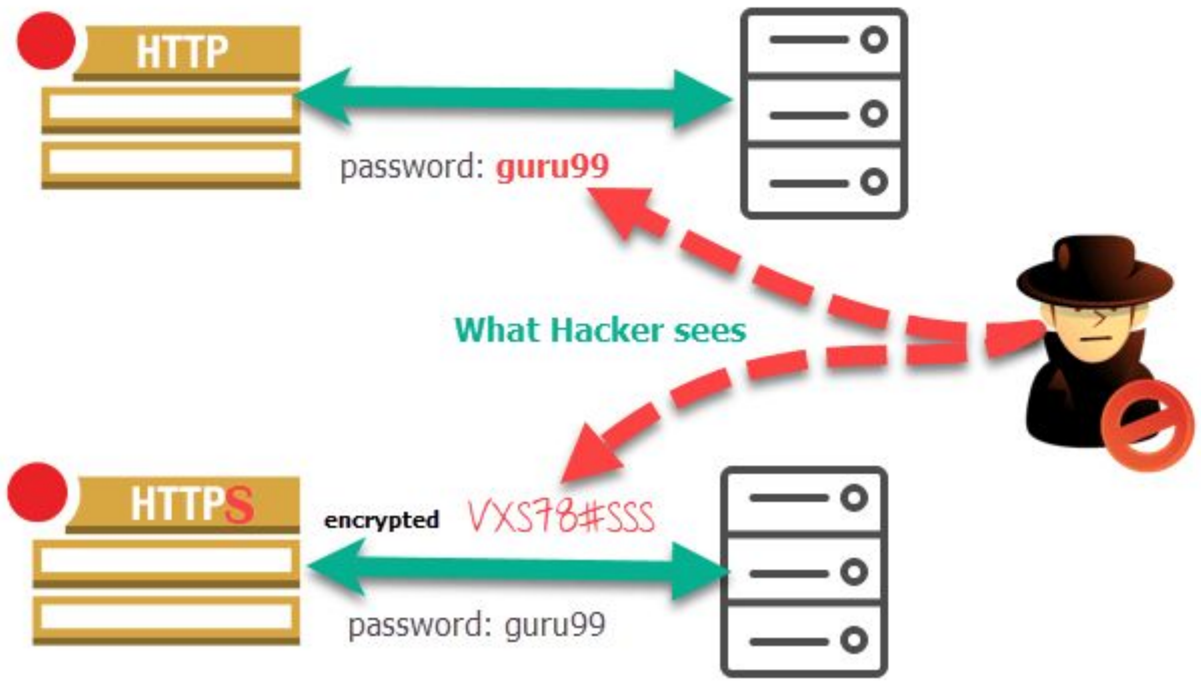
HTTP vs HTTPS

HTTP sends all data as plain text that is readable by anyone.

Like mailing a letter on a postcard.

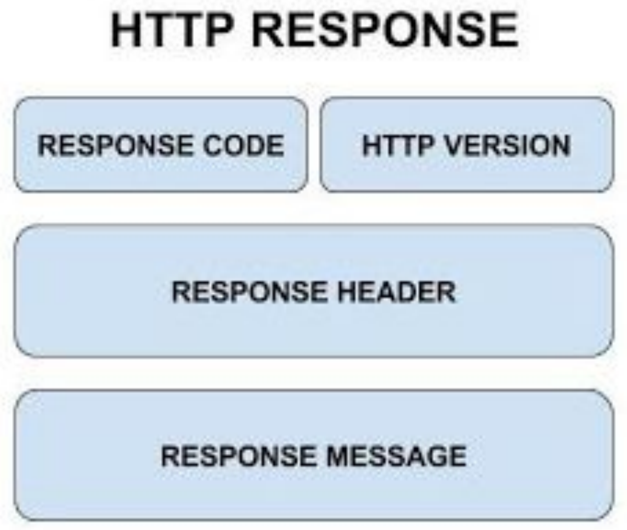
HTTPS encrypts data using TLS (Transport Layer Security), so data is not readable.

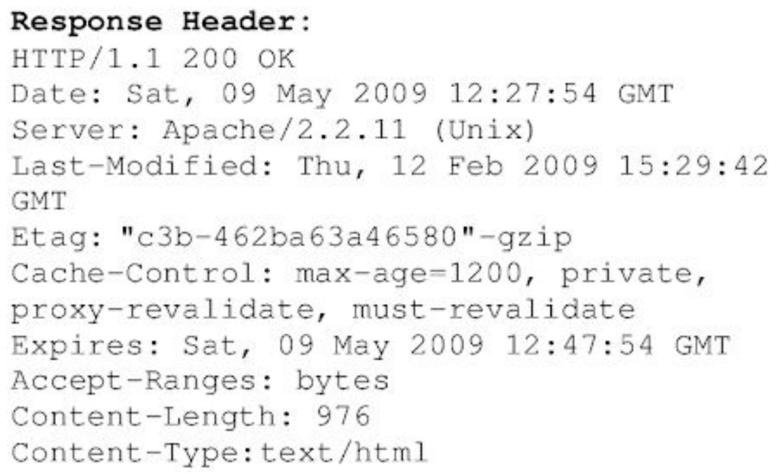
Like mailing a letter in a sealed envelope.



HTTP Response

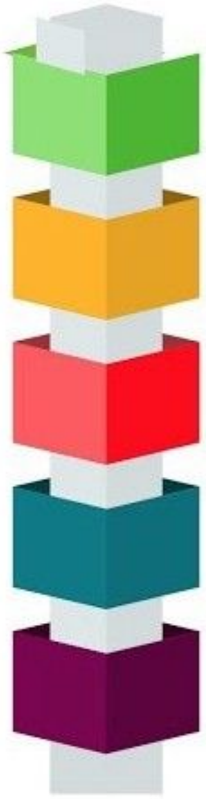
1. **Status Code**
   The HTTP response status code indicates whether or not the request was successful.
2. **Header**
   Meta Information about the response.
3. **Content Type**
   The type of content being returned: text, json, image, stream, etc.
4. **Content (Body)**
   The data of the response. HTML, image, data, video stream, etc.





```
Response Header:
HTTP/1.1 200 OK
Date: Sat, 09 May 2009 12:27:54 GMT
Server: Apache/2.2.11 (Unix)
Last-Modified: Thu, 12 Feb 2009 15:29:42
GMT
Etag: "c3b-462ba63a46580"-gzip
Cache-Control: max-age=1200, private,
proxy-revalidate, must-revalidate
Expires: Sat, 09 May 2009 12:47:54 GMT
Accept-Ranges: bytes
Content-Length: 976
Content-Type:text/html
```

HTTP Response Status Code



Status Code	The Server is Saying...
1XX INFORMATIONAL	**I need to tell you something**
2XX SUCCESS	**Everything is OK.** 200-OK 201-Created
3XX REDIRECTION	**You need to go somewhere else for that**
4XX CLIENT ERROR	**You messed up.** 400-Bad Request 401-Unauthorized 403-Forbidden 404-Not Found
5XX SERVER ERROR	**I messed up.** 500 - Server Exception

418 I'm a teapot

The HTTP `418 I'm a teapot` client error response code indicates that the server refuses to brew coffee because it is, permanently, a teapot. A combined coffee/tea pot that is temporarily out of coffee should instead return 503. This error is a reference to Hyper Text Coffee Pot Control Protocol defined in April Fools' jokes in 1998 and 2014.

Some websites use this response for requests they do not wish to handle, such as automated queries.

Status

418 I'm a teapot

Copy to Clipboard

Specifications

Specification
RFC 2324 # section-2.3.2

API

- Application Programming Interface (API) is set of functions and/or procedures designed to interact with an external system.
- Modern cloud architecture relies heavily on API’s.
- Consuming an API means interacting with an API’s code to produce a desired result.
- Most modern API's use the Representational state transfer (REST) model implemented via the HTTP Protocol
- API's or services that use the REST model are said to be RESTful Web Services (more on this next).

API Calls

- Make some API Calls Using Postman
- Try out a public API on the Web

RESTful

- REST (REpresentational State Transfer) is a software architectural pattern that defines a set of constraints and standards for how applications on a network should communicate.
- In other words, REST describes a set of rules that specifies the best practices for how communication on the web should work.

Rules of REST

1. Uniform Interface (URI, URL)
2. Stateless
3. Cacheable
4. Client-Server based
5. Layered System

- Most commonly uses HTTP(S) as the application protocol and JSON as a response.
- Also commonly referred to as REST API, REST Web API, or REST Web Service
- Use nouns instead of verbs for resource endpoints.
- Name collections with plural nouns (e.g. hotels).
- Use nested resource endpoints to show relationships (e.g. `hotels/10/reservations`).

JSON

- JSON (JavaScript Object Notation): messaging format used by RESTful Web Services.
- Lightweight data-interchange format
- Easy for humans to read & write
- Easy to parse in code
- Text Only/Language Independent
- Often used for passing data around the internet
- Made up of name/value pairs
- Name is always a quoted string (i.e. "address")
- Name/value separated by :
- Value can be one of several types

Number	Any number (integer or decimal)
String	A sequence of zero or more Unicode characters. Strings are delimited with double-quotation marks and support a backslash escaping syntax.
Boolean	Either of the values true or false
Array	An ordered list of zero or more values, each of which may be of any type. Arrays use [] square bracket notation with comma-separated elements.
Object	A collection of name–value pairs where the names (also called keys) are strings. Objects are delimited with {} curly braces and use commas to separate each pair, while within each pair the colon ':' character separates the key or name from its value.
null	

JSON

JSON (JavaScript Object Notation) is a messaging format that is commonly used by **RESTful** Web Services.

JSON uses key/value pairs to represent objects.

Objects are identified by **{ }**, Arrays by **[ ]**

Objects have **properties** and **values** separated by a **colon**.   The property names are Strings in double quotes. The value can be a number, String (with double quotes), Array, or Object.

Each property/value pair, object, or array is separated by a **comma**.

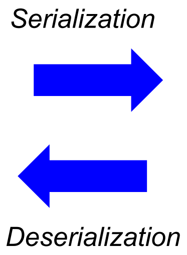
```
{
  "reviews": [
    {
     "hotelID": 1,
     "title": "What a great hotel!",
     "review": "I thought this was a really great
hotel and would stay again!",
     "author": "John Smith",
     "stars": 4
    },
    {
     "hotelID": 1,
     "title": "Peaceful night sleep",
     "review": "I had a really good night sleep
and would stay again",
     "author": "Kerry Gold",
     "stars": 3
    }
  ]
}
```

JSON

Java Objects can easily be converted to JSON data and vice versa, Mapping from an Object to JSON is known as serialization of an Object. Mapping from JSON to an Object is known as deserialization.

```
public class Review {
    private int hotelID;
    private String title;
    private String review;
    private String author;
    private int stars;
}

Review[] hotelReviews;
```



```
[
    {
        "hotelID": 1,
        "title": "What a great hotel!",
        "review": "Great hotel,
        "author": "John Smith",
        "stars": 4
    },
    {
        "hotelID": 1,
        "title": "Peaceful night sleep",
        "review": "Would stay again",
        "author": "Kerry Gold",
        "stars": 3
    }
]
```

- A module of the Spring Framework that focuses on microservices and allows building stand-alone applications with minimal or no configuration.

- Widely used to consume or build RESTful services.

- Abstracts much of the repetitive and tedious work needed to create or use a Web Service.

- Contains an embedded application/web server (Tomcat).

Service

- A service class encapsulates the functionality of code that works as a unit or provides a specific business purpose. This allows other code in the application to interact with the Service Class without needing to know or understand the inner workings of the functionality.
- For example: An API service class would encapsulate the functionality of an API so the other code could interact with it as a business unit rather than needing to understand the functionality of the API.

GET

The `RestTemplate` class provides the means with which we can make a request to an API. Here is an example call:

```
RestTemplate restTemplate = new RestTemplate(); // Create a new client
ResponseEntity response =  restTemplate.getForEntity(
                "https://api.exchangerate-api.com/v4/latest/USD",
                String.class); // Make GET request using Client

//return data returned from .getBody()
System.out.println(response.getBody());
```

`response.getBody()` returns a `String` representation of the JSON, just like if you saw the API response in your browser. In the next slide, you'll see how `RestTemplate` can automatically convert the response data into a Java Object.

GET

The `RestTemplate` class provides a way to make a request to an API and treat the result as a Java Object. The JSON returned by the API call will be mapped to the specified Java class. Here is an example call:

```
private static final String API_BASE_URL =
    "http://helpful-site/v1/api/objects/1";

private static RestTemplate restTemplate = new RestTemplate();
MyObj myobj = restTemplate.getForObject(API_BASE_URL, MyObj.class);
```

Note that we can specify the return type of the API call with the second parameter (`MyObj.class`). Alternatively, if you are getting an array of objects back, we can write the following:

```
MyObj[] myobj = restTemplate.getForObject(API_BASE_URL, MyObj[].class);
```

HTTP Methods

We saw GET, which simply reads the data. Let's look at other request types that might potentially change the application’s data:

- POST:
  - Indicates that the request will add new data to the the server.
  - Post modifies the server and leaves the server in a different state each time the same request is repeated.
  - Ideally suited for inserting new data into the data source.
- PUT:
  - Indicates that the request will update existing data on the the server.
  - Put changes the state on the server but the state remains the same if the same request is repeated multiple times.
  - Ideally suited for updating an existing record within a data source.
- DELETE:
  - Indicates that the request will remove existing data from the the server.
  - Ideally suited for removing an existing record from the data source.

GET, POST, PUT & DELETE

GET	http://localhost:3000/reservations
POST	http://localhost:3000/reservations
PUT	http://localhost:3000/reservations/{id}
DELETE	http://localhost:3000/reservations/{id}

Idempotent

Idempotent means that a request that has the same result regardless of how many times it is completed.

- GET, PUT, DELETE are idempotent, meaning that calling it once or several times successively has the same effect (that is no side effect).
- POST is NOT idempotent - we usually use it for creating data. which means each call could wind up with a different effect (i.e. create a different set of data).

POST

Create HTTP Headers for POST

Set the **content-type** for JSON

```
private static final String API_BASE_URL = "http://localhost:3000/";
private final RestTemplate restTemplate = new RestTemplate();

public Reservation addReservation(Reservation newReservation) {
  HttpHeaders headers = new HttpHeaders();
  headers.setContentType(MediaType.APPLICATION_JSON);

  HttpEntity<Reservation> entity = new HttpEntity<Reservation>(newReservation, headers);

  Reservation savedReservation =
    restTemplate.postForObject(API_BASE_URL + "reservations", entity, Reservation.class);

  return savedReservation;
}
```

Create an **HttpEntity**, which allows us to combine headers and body

Call **postForObject** with the **HttpEntity** and class to post for (Reservation).

PUT

- PUT requests are similar to POST requests in that they usually have both headers and a payload contained in the message body.
- We can write code for a PUT request much like our POST code but using the `put()` method rather than `postForObject()` method.

```
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_JSON);

HttpEntity<Reservation> entity =
  new HttpEntity<Reservation>(updatedReservation, headers);

restTemplate.put(API_BASE_URL + "reservations/" + updatedReservation.getId(), entity);
```

DELETE

- DELETE requests are similar to GET requests In that they have only headers and not a payload contained in the message body.
- We can write code for a DELETE request much like our GET code but using the `delete()` method rather than `getForObject()` method.

```
restTemplate.delete(API_BASE_URL + "reservations/" + id);
```

Exceptions

There are 2 exceptions to be aware of when dealing with APIs:

- `RestClientResponseException` - thrown for code other than a 2XX
    - Can check status code via this exception's `getRawStatusCode()` method
    - Can get text description of the status code (i.e. Not Found for 404) from this exception's getStatusText() method
- `ResourceAccessException` - is thrown when there was a network issue that prevented a successful call.

```java
try {
     Reservation savedReservation = restTemplate.postForObject(API_BASE_URL +
          "reservations", entity, Reservation.class);
     return savedReservation;
} catch (RestClientResponseException ex) {
     BasicLogger.log("Error: " + ex.getRawStatusCode() + " " + ex.getStatusText());
} catch (ResourceAccessException ex) {
     BasicLogger.log("Error: " + ex.getMessage());
}
```

HTTP Status Code

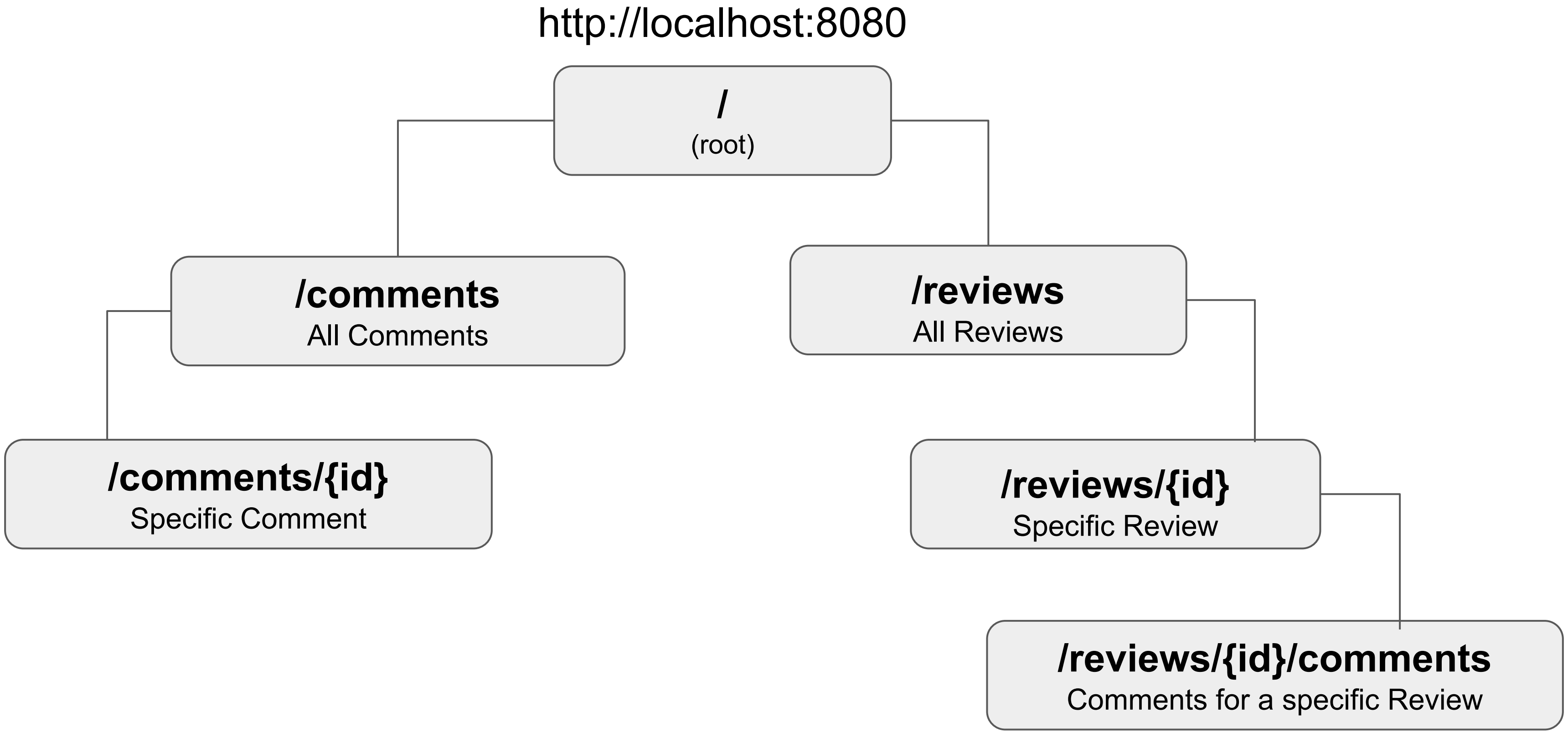
HTTP Status Text

RESTful Service Design - End Points

Designing endpoints

1. Endpoints should be plural names that define the entity (what is being accessed)
    a. Hotels, Reservations, Cities, etc.
2. IDs should be sent as Path Variables
3. Filters and Sorting should be sent as Query Parameters
4. Endpoints should reflect the hierarchy of the entities

Endpoint Hierarchy - Reviews



Example Endpoint Design for Reviews

Endpoint	Method	Description	Success	Error
/api/reviews	GET	List All Reviews	List of Reviews	
/api/reviews	POST	Add a Review	201 - CREATED	
/api/reviews/{id}	GET	Get a specific Review	Review	404 - Review Not Found
/api/reviews/{id}	PUT	Update a Review	200 - OK	404 - Review Not Found
/api/reviews/{id}	DELETE	Delete a Review	204 - No Content	404 - Review Not Found
/api/reviews/{id}/comments	GET	List Comments for a Review	List of Comments	404 - Review Not Found
/api/reviews/{id}/comments	POST	Add a comment to a Review	201 - CREATED	404 - Review Not Found
/api/comments/{id}	GET	Get a specific Comment	Comment	404 - Comment Not Found
/api/comments/{id}	PUT	Update a Comment	200 - OK	404 - Comment Not Found

With API week coming up next week, here are some free public APIs you may enjoy sharing with your students:

HTTP://API.SHOUTCLOUD.IO/V1/SHOUT (good basic demo that switches anything you POST to it into ALL CAPS)
http://deckofcardsapi.com

Two "reflective" APIs that are good for testing HTTP requests of various types:
https://httpbin.org
https://httpstat.us/

Fun (Pokemon, Star Wars, Harry Potter)
https://pokeapi.co
https://swapi.dev/
https://wizard-world-api.herokuapp.com/swagger/index.html

Potentially useful for side projects or capstones
https://date.nager.at/Api (list public holidays for any country and year)
https://api.dictionaryapi.dev/api/v2/entries/en/<word>
https://quickchart.io/documentation/ (create charts and graphs from POST'd data)
https://goqr.me/api/doc/create-qr-code/ (create QR codes for any text or url)

All of those APIs are open and don't require any kind of key or authentication. A couple more free APIs with interesting data that require registering for a free key are:
https://www.nps.gov/subjects/developer/api-documentation.htm (National Park Service)
https://fdc.nal.usda.gov/api-guide.html (USDA food data)

Important Vocabulary

client: software that sends a request to a server to access a shared resource and processes the response returned from the server. (web browser, phone, pgadmin)

server: generic term referring to either software or hardware that processes request from clients (web server, email server, postgreSQL db)

Know the definition and structure of requests and responses

Know the difference between stateless and stateful

Know these request methods:

- GET - Retrieves information
- POST - Adds information
- PUT - Updates information
- DELETE - Deletes information

Know these response status code ranges:

- 100-199 - Informational responses
- 200-299 - Successful responses
- 300-399 - Redirection messages
- 400-499 - Client error responses
- 500-599 - Server error responses

URL: tells a client how to make a request to a server for a specific resource.

IP address- identifies a computer or device on a network (including the internet)

DNS: a service service allows for easy to remember names for humans to use (converts domain names to ip addresses)

port: a process-specific or an application-specific software construct serving as a communication endpoint

Important Vocabulary Continued

Know the difference between HTTP and HTTPS

Basic web page request workflow:

API (Application Programming Interface): a way for software components to talk to each other.

Web Service/Web API: an API that allows for applications to communicate with each other across the internet

REST(REpresentational State Transfer): A software architectural pattern that defines a set of constraints and standards for how 2 applications on a network should communicate.

JSON: (JavaScript Object Notation) is a messaging format that is commonly used by RESTful Web Services.