Web Service

* SOA (Service Oriented Architecture)
  + Service is going to be a **black box** API (application program interface) for applications and other services to use
  + Service is not for humans to consume directly
    - They usually receive and send responses in non-human friendly formats
      * JSON, XML
  + Web Applications are what humans can use
    - You have a GUI front end
    - Interacts with services on the backend
* A Web Service is a service available over the web, aka the internet

Why Web Services?

* The black box (You do not know how it works) nature of web services increases interoperability
  + You can write a service in any language and have it communicate with another service
  + Makes it easy to add new services
* Services separate GUI human application code from the logical CRUD interactions that occur in the service
  + Removes the old-school idea of server side rendering

REST (REpresentation State Transfer)

* REST is a type of Web Service
* A web service that conforms the REST constraints is considered to be RESTful and therefor a REST Web Service
* REST Constraints
  1. Client-server architecture
     + Separation of concern from the UI GUI layer (client) from the data processing and resource management Server layer
     + You should be able to make a RESTful web service and have two different front-ends and no one could notice
  2. Statelessness
     + REST web services do NOT maintain the USER session
     + Sessions tie the web service to the client side
  3. Layered System
     + You can stack RESTful web services on top of each other
     + AuthorizationService => UserService
  4. Cacheable
     + You can choose to store information on either the client or server side
     + Ex. If your application notices everyone is requesting a certain record in the database you may choose to store it temporarily on the server. Decreases the time it would take to get that information from the database
  5. Code-on-demand(Optional)
     + Most of the time a service will send back JSON, XML or some other data format
     + You can\* send back executable code
       1. Send back JavaScript snippets for the browser to run
  6. Uniform interface
     + Standard set of conventions for creating your API (URL endpoints)
       1. Identification of resources through the URI
          - A resource is going to be any collection of object
          - Ex you have a zoo
          - [www.zoo.com/animals](http://www.zoo.com/animals) => Returns all animals
          - [www.zoo.zom/animals/3](http://www.zoo.zom/animals/3) =>retrurns the animal with an id of 3
          - [www.zoo.com/employees](http://www.zoo.com/employees) => Returns all employes
          - [www.zoo.com/employees?name=smith](http://www.zoo.com/employees?name=smith)

Queries employees for someone named smith

Should return a list of employees who have the name smith

* + - 1. Manipulation of Resources through representations
         * A single resource can be represented in multiple formats
         * I should be able to get an employee in JSON, or XML, or plain text, or JPEG
         * Those are all representations of the resource
      2. Self descriptive messages
         * Each request you make to a REST web service had all the information you need to process it
         * You should NOT have to be dependent on another request
         * EX you send one request with the API key
         * NOT one request that has an API key to set up a session that then validates a following request
      3. Hypermedia(hyperlinks) As the Engine Of The Application State (HATEOAS)
         * When you make a request to a resource you should get back url links to related resources
         * [www.zoo.com/animals](http://www.zoo.com/animals) => returns all animals

Returns a refence to [www.zoo.com/habitats](http://www.zoo.com/habitats) which tells you where the animals live in the zoo

* REST technically does not require documentation
* But please write documentation