REST

(REpresentational State Transfer)

REST Vs SOAP

- Simple web service as an example: querying a phonebook application for the details of a given user

- Using Web Services and SOAP, the request would look something like this:

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
   soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
        <soap:body pb="http://www.acme.com/phonebook">
                <pb:GetUserDetails>
                        <pb:UserID>12345</pb:UserID>
                </pb:GetUserDetails>
        </soap:Body>
</soap:Envelope>
```

REST Vs SOAP

- Simple web service as an example: querying a phonebook application for the details of a given user

- And with REST? The query will probably look like this: http://www.acme.com/phonebook/UserDetails/12345

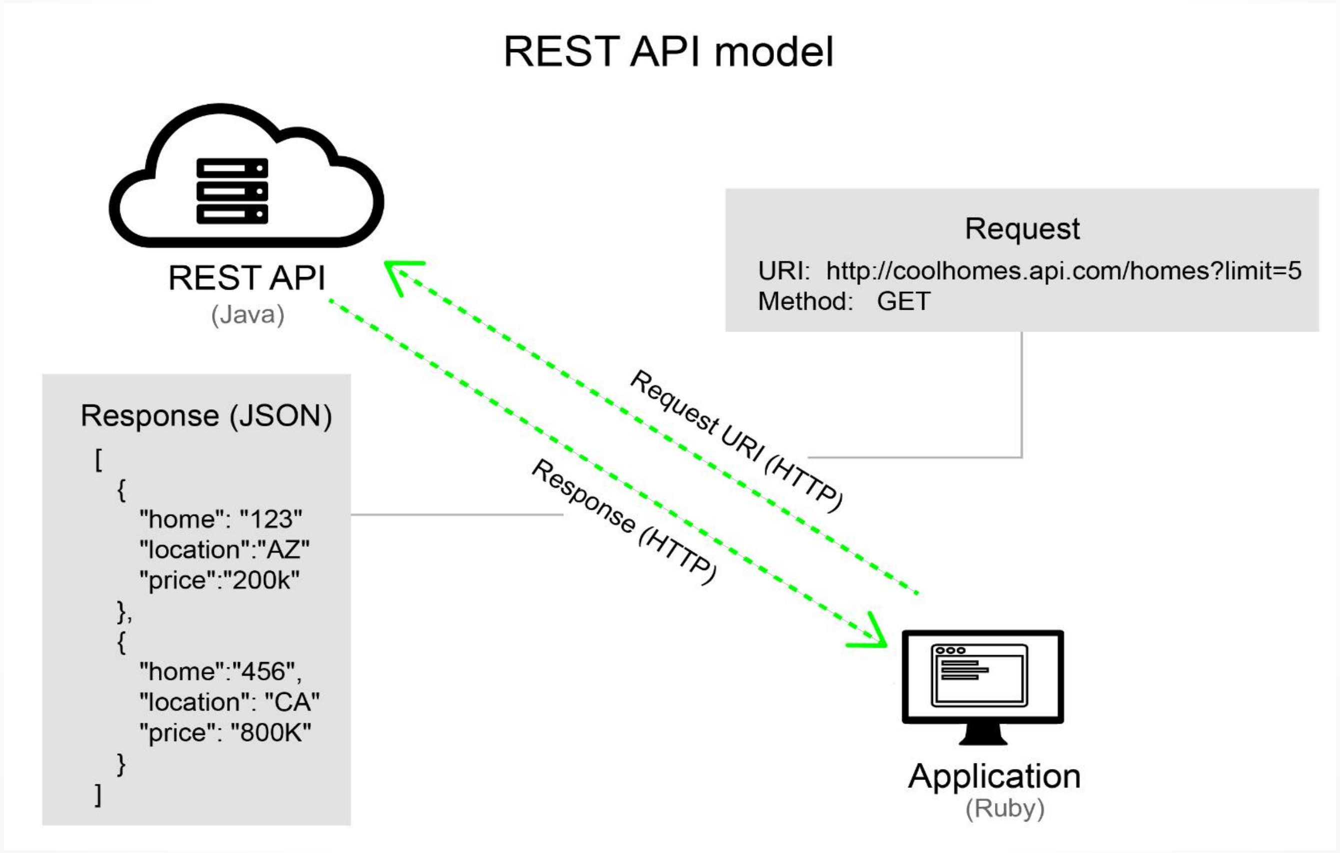
- GET /phonebook/UserDetails/12345 HTTP/1.1
  Host: www.acme.com
  Accept: application/xml

- Complex query:
  http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe

What is REST?

- The REST architecture was originally designed to fit the HTTP protocol that the world wide web uses.
- Central to the concept of RESTful web services is the notion of resources.
- Resources are represented by URIs.
- The clients send requests to these URIs using the methods defined by the HTTP protocol, and possibly as a result of that the state of the affected resource changes.

ReST API Model



Request
URI: http://coolhomes.api.com/homes?limit=5
Method: GET

Response (JSON)

```
[
  {
    "home": "123"
    "location":"AZ"
    "price":"200k"
  },
  {
    "home":"456",
    "location": "CA"
    "price": "800K"
  }
]
```

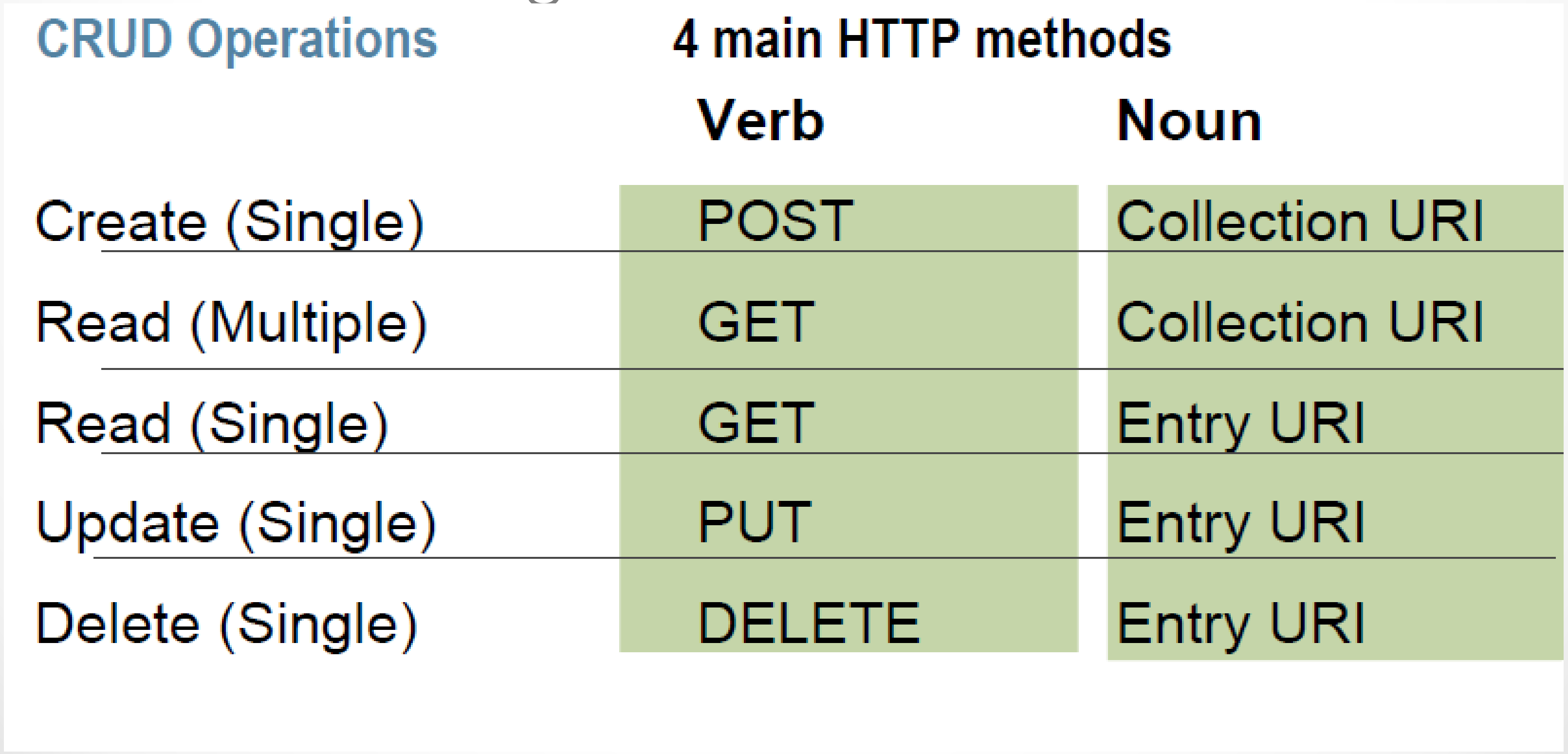
http://www.javapassion.com/webservices/RESTPrimer.pdf

JSON

- JSON (**J**ava**S**cript **O**bject **N**otation)
  - Data is in name/value pairs
  - Data is separated by commas
  - Curly braces hold objects
  - Square brackets hold arrays
  - Value can be a string, number, object, array, boolean or null

REST over HTTP – Uniform interface

- CRUD operations on resources
  - Create, Read, Update, Delete
- Performed through HTTP methods + URI



CRUD Operations	4 main HTTP methods	
	Verb	Noun
Create (Single)	POST	Collection URI
Read (Multiple)	GET	Collection URI
Read (Single)	GET	Entry URI
Update (Single)	PUT	Entry URI
Delete (Single)	DELETE	Entry URI

HTTP Methods

HTTP Method	Action	Examples
GET	Obtain information about a resource	http://manipal.edu/api/v1/students (retrieve student list)
GET	Obtain information about a resource	http://manipal.edu/api/v1/students/123 (retrieve student #123)
POST	Create a new resource	http://manipal.edu/api/v1/students (create a new student, from data provided with the request)
PUT	Update a resource	http://manipal.edu/api/v1/students/123 (update student #123, from data provided with the request)
DELETE	Delete a resource	http://manipal.edu/api/v1/students/123 (delete student #123)

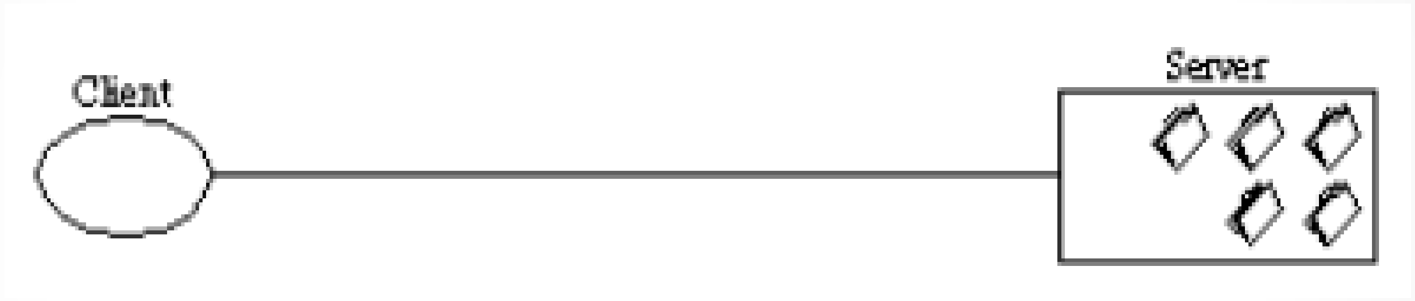
HTTP Status Codes

Level 200 (Success)	Level 400	Level 500
200 : OK	400 : Bad Request	500 : Internal Server Error
201 : Created	401 : Unauthorized	503 : Service Unavailable
203 : Non-Authoritative Information	403 : Forbidden	501 : Not Implemented
204 : No Content	404 : Not Found	504 : Gateway Timeout
	409 : Conflict	599 : Network timeout
		502 : Bad Gateway

Characteristics of a ReST system

- Client-Server

  Servers and clients may also be replaced and developed independently, as long as the interface between them is not altered.



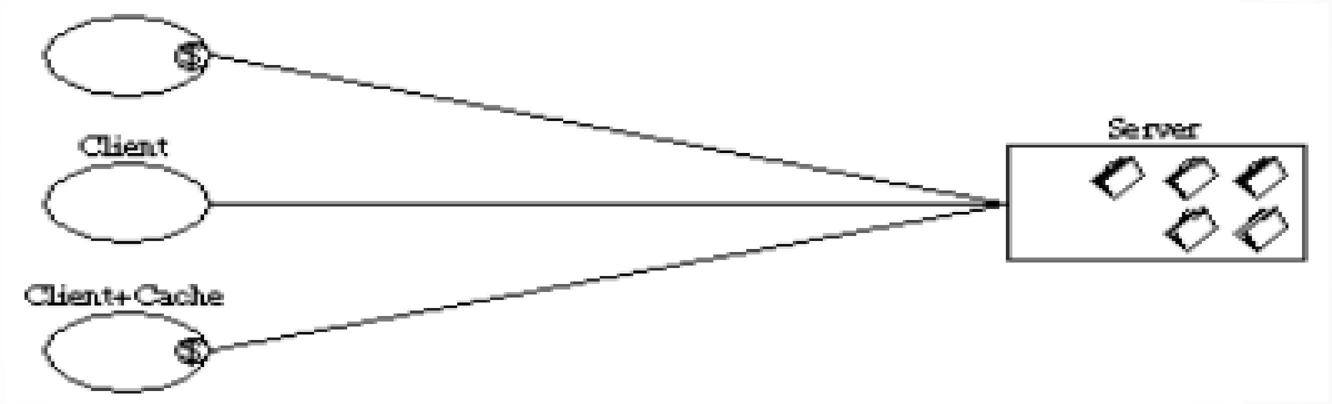
- Stateless

  No client context shall be stored on the server between requests. Client is responsible for managing the state of application.

Characteristics of a ReST system

- Cacheable

  Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance.



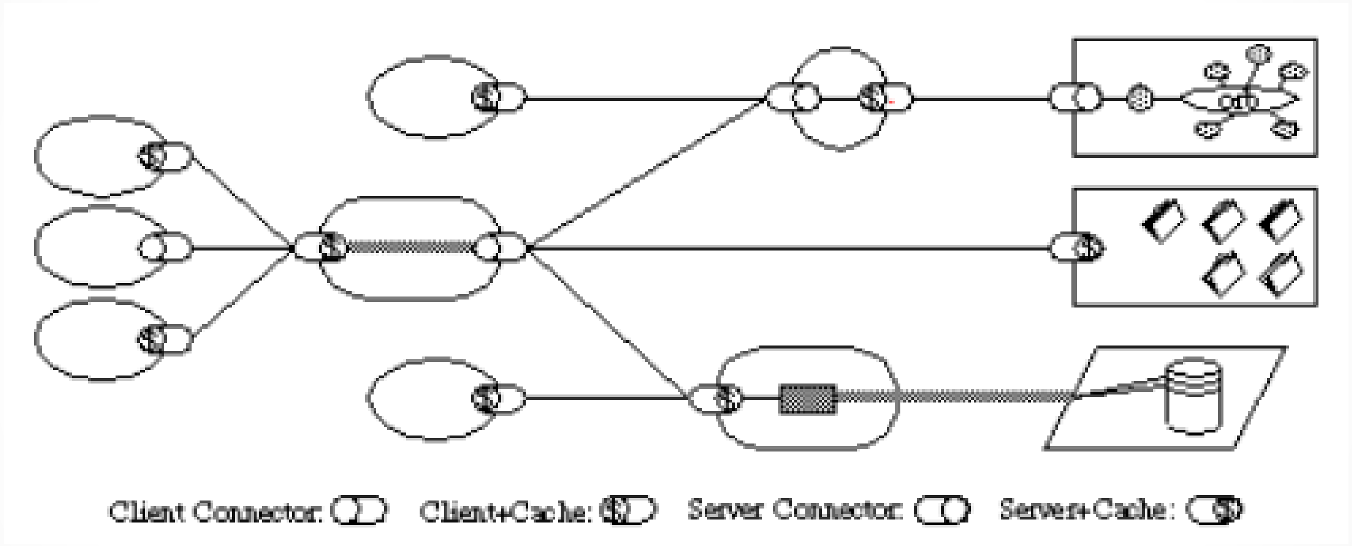
- Uniform Interface

  Once a developer become familiar with one of your API, he should be able to follow similar approach for other APIs.

Characteristics of a ReST system

- Layered System

  Deploy the APIs on server A, and store data on server B and authenticate requests in Server C



- Code on Demand

  Clients may call your API to get a UI widget rendering code

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Characteristics of a REST system

- **Client-Server**: A Client-Server Architecture separates the user interface (the Client) from data processing and storage (the Server) so that a system has improved portability, scalability, and malleability (ability to modify on the fly).

- **Stateless**: Each request from a client must contain all the information required by the server to carry out the request. In other words, the server cannot store information provided by the client in one request and use it in another request. This makes the interface between Clients and Servers reliable.

- **Cacheable**: Caching is all about making information available at the right place at the right time. Your web browser caches data from websites (assets like images, colors, types, fonts, etc) so that it can reload them faster the next time you view them. Caching, if implemented properly, can improve the scalability and performance of an application significantly.

Characteristics of a REST system

- **Uniform Interface**: The uniform interface constraint is fundamental to the design of any RESTful system. It simplifies and decouples the architecture, which enables each part of a system (e.g. the Client or the Server) to evolve independently – which makes a system more flexible and easier to maintain..

- **Code on demand**: Code on Demand means that Servers can temporarily extend or customize the functionality of a Client by transferring executional code like Java applets or client-side JavaScript. This increases the malleability of an application built on a REST API.

REST API Design

- Resources form the nucleus of any REST API design. Resource identifiers (URI), Resource representations, API operations (using various HTTP methods), etc. are all built around the concept of Resources. It is very important to select the right resources and model the resources at the right granularity while designing the REST API so that the API consumers get the desired functionality from the APIs, the APIs behave correctly and the APIs are maintainable.
- A resource can be a singleton or a collection. For example, “customers” is a collection resource and “customer” is a singleton resource (in a banking domain). We can identify “customers” collection resource using the URN “/customers”. We can identify a single “customer” resource using the URN “/customers/{customerId}”.
- A resource may “contain” sub-collection resources also. For example, sub-collection resource “accounts” of a particular “customer” can be identified using the URN “/customers/{customerId}/accounts” (in a banking domain). Similarly, a singleton resource “account” inside the sub-collection resource “accounts” can be identified as follows: “customers/{customerId}/accounts/{accountId}”.

REST API Design (contd.)

- The starting point in selection of resources is to analyze your business domain and extract the nouns that are relevant to your business needs. More importantly, focus should be given to the needs of API consumers and how to make the API relevant and useful from the perspective of API consumer interactions. Once the nouns (resources) have been identified, then the interactions with the API can be modeled as HTTP verbs against these nouns. When they don't map nicely, we could approximate. For example, we can easily use the "nouns in the domain" approach and identify low level resources such as Post, Tag, Comment, etc. in a blogging domain. Similarly, we can identify the nouns Customer, Address, Account, Teller, etc. as resources in a banking domain.
- If we take "Account" noun example, "open" (open an account), "close" (close an account), "deposit" (deposit money to an account), "withdraw" (withdraw money from an account), etc. are the verbs. These verbs can be nicely mapped to HTTP verbs. For example, API consumer can "open" an account by creating an instance of "Account" resource using HTTP POST method. Similarly, API consumer can "close" an account by using HTTP DELETE method. API consumer can "withdraw" or "deposit" money using HTTP "PUT" / "PATCH" / "POST" methods

- Source: http://www.thoughtworks.com/insights/blog/rest-api-design-resource-modeling

Testing ReST using PostMan

- Create a new request
- Paste the url
- Set the appropriate HTTP Method
- In Header set Content-Type as application/json
- In case of POST, the json can be sent to the server in body with selection as raw.
-