DEVELOPING WEB SERVICES

ReST

REST

REpresentational State Transfer

- ReST is a simple way to organize interactions between independent systems.
  - REpresentational State Transfer
- Resources Based
  - Identified by URI
    - Multiple URIs can refer to the same resource
  - The representation is different then the resource
- It is a design pattern that uses an architectural design
- Any format are supported - typically json or xml

HTTP REQUESTS

https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods

- Standard verbs used to denote what action should be taken

  GET

  POST

  PUT

  DELETE

  HEAD

  TRACE

  OPTIONS

REST

Verbs / Resources

HTTP Method	URI	Operation
GET	/users/	Get list of users
DELETE	/users/1	Delete User with Id 1
GET	/users/1	Get User with Id 1
POST	/users/2	Insert User with Id 2
PUT	/users/2	Update User with Id 2

REST PRINCIPALS

Constraints applied to the architecture

- The REST architectural style describes six constraints.
- These constraints define the basis of RESTful-style
  - Uniform Interface
  - Stateless
  - Cacheable
  - Client-Server
  - Layered System
  - Code on Demand (optional)
- If a service violates any other constraint, it cannot strictly be referred to as RESTful.

UNIFORM INTERFACE

ReST Principals / Constrains

- Defines the interface between client and server, simplified and decouples the architecture.
  - Resource based
    - Anything can be a resource
    - Resources themselves are separate from the representations that are returned to the client
- Manipulation of Resources Through Representations
  - If a client has a representation of a resource, they can manipulate that resource (if allowed)
- Self-descriptive Messages
  - Message have enough information to describe how it should be processed.
- Hypermedia as the engine of application state (HATEOAS)
  - Resource should contain links pointing to URIs to fetch related information

STATELESS

ReST Principals / Constrains

- All Client/Server communications are stateless
- Each request from the client is independent and should contain all information required to handle the request
- Enables greater scalability since the server does not have to maintain, update or communicate that session state.
- The server does not store anything

CACHEABLE

ReST Principals / Constrains

- Clients can cache responses.
- Responses must define themselves as cacheable to prevent clients reusing stale or inappropriate data
- Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance.
- The **Cache-Control** (in the header response) determines if the response should be cached or not and for how long

CLIENT / SERVER

ReST Principals / Constrains

- Servers and Clients have different concerns
- Servers store and handle information and makes it available to clients in an efficient manner.
- Clients takes that information and displays it to the user or they uses it to perform other requests for information.
- The client and server MUST be able to evolve separately without any dependency on each other

LAYERED SYSTEM

ReST Principals / Constrains

- There can be multiple layers in between client and server
- Layers can also help caching - Intermediary
  - Increased Performance and scalability
- Layers can include proxies - Client Side
- Gateways - Server Side

CODE ON DEMAND

ReST Principals / Constrains

- This is optional
- Ability to transfer logic to the client side
  - functionality is extended by downloading and executing code in the form of applets or scripts
  - Simplifies clients by reducing the number of features required to be pre-implemented

HTTP REQUESTS

Examples

HTTP Method	URI	Operation	Operation Type
GET	/users/	Get list of users	Read Only Idempotent + Safe
GET	/users/1	Get User with Id 1	Read Only Idempotent + Safe
POST	/users/2	Insert User with Id 2	Not Idempotent Not Safe
PUT	/users/2	Update User with Id 2	Idempotent (Usually) Not Safe
DELETE	/users/1	Delete User with Id 1	Idempotent (Usually) Not Safe

- **SAFE** - No risk of data modification or corruption
  - Calling it once = calling it 10 times = never calling it
- **IDEMPOTENT**
  - Calling multiple identical requests ends up having the same result as a single request

REST

CRUD examples

C reate

```
POST /user/ HTTP/1.1
Host: stephmoreau.ca
Content-type: application/x-www-form-urlencoded

firstName=John&lastName=Smith&country=Canada
```

R ead

```
GET /user/1125 HTTP/1.1
Host: stephmoreau.ca
Content-type: application/x-www-form-urlencoded
```

U pdate

```
PUT /user/1125 HTTP/1.1
Host: stephmoreau.ca
Content-type: application/x-www-form-urlencoded

firstName=John&lastName=Smith&country=Canada
```

D Elete

```
DELETE /user/1125 HTTP/1.1
Host: stephmoreau.ca
Content-type: application/x-www-form-urlencoded
```

HTTP STATUS CODES

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

- 1xx - Information Base (Information about your request)
    - Request was received and understood. client to wait for a final response.
- 2xx - Success
    - Action requested by the client was received, understood and accepted
- 3xx - Redirection
    - Client must take additional action to complete the request.
    - User agent may carry out the additional action with no user interaction only depending on the method
- 4xx - Client Error
    - Error produces that seems to have been caused by the client
- 5xx - Server Error
    - The server failed to fulfil a request