GREAT INDIAN DEVELOPER SUMMIT™
April 25-28, 2017. Bangalore
Is REST - the BEST?

Birju Shah - 25/04/2017 - #GIDS17
Everything small businesses need to fuel their online presence and reach customers everywhere.
Objective

“We shall discuss the current state of RESTful APIs, and what we originally intend to do with APIs. By the end of the session we shall try to come up with efficient ways to construct and consume APIs”
Agenda

Remote API

RESTful API

Context in API Design

REST Characteristics

REST Ambiguities

Alternatives and benchmarks

Conclusion
API

“A set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or any service.”
Remote API

- Web Service Host & Port
- Locate a Method
- Input parameters with type
- Build Input (For ex. JSON, XML ..)
- Mode of transport (For ex. HTTP)
- How to initiate execution of API
- Output with type
- Efficiency
REST

“An architectural style of networked systems”

“Not a protocol nor a standard”

“Think WEB as REST”

“No practical implementation as Reference”

-- Roy Fielding
Representational state transfer

http://api.aircraft.com/aircrafts/

http://api.aircraft.com/aircrafts/boeing_737

- Fuel Requirements
- Cost
- Availability
http://img.com/boing_737.jpeg
REST Characteristics & Constraints

- Client-Server
- Stateless
- Cache
- Uniform Interfaces
- Interoperability
- Interconnected Resources
- Named Resources (URI)
Richardson Maturity Model

- Level 0: The Swamp of POX
- Level 1: Resources
- Level 2: HTTP Verbs
- Level 3: Hypermedia Controls

Glory of REST

Reference: https://martinfowler.com/articles/richardsonMaturityModel.html
# RESTful API

## HTTP Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>URL Template</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>api/profile/{ID}</td>
<td>Returns resource representation.</td>
</tr>
<tr>
<td>POST</td>
<td>api/profile</td>
<td>Creates new resource. New URL is returned in Location header. Response body is optional. Often used for business operations not easily mapped other HTTP methods.</td>
</tr>
<tr>
<td>PUT</td>
<td>api/profile/{ID}</td>
<td>Updates or creates new resource. Response body is optional.</td>
</tr>
<tr>
<td>DELETE</td>
<td>api/profile/{ID}</td>
<td>Deletes resource.</td>
</tr>
<tr>
<td>HEAD</td>
<td></td>
<td>Same as GET but no body is returned.</td>
</tr>
<tr>
<td>OPTIONS</td>
<td></td>
<td>Information on what methods are supported.</td>
</tr>
</tbody>
</table>
REST Ambiguities

- HTTP verbs & Methods does not comply with all actions / operations.
- Error Handling
- Noun Vs Verbs (I care about calling remote method)
- API Should not be limited / restricted by HTTP Semantics
- JSON message formats, needs to be parsed and converted to entities on client & server.
- No W3 standard
- Fuzzy for non CRUD Operations. Custom Headers to satisfy some requirements.
- Versioning of APIs (Headers / URL, Timestamp / Version number, Backward compatibility)
- Confusion arises to represent property or relationship between resources.
- Errors & Retry logic (A client’s headache)
- Timeout (HTTP headache)
- Typesafe (Developer’s headache)
- Include Types in JSON
- Validation Rules for Javascript client
REST Ambiguities (Error Handling)

Google GData

| Status Codes | 200 201 304 400 401 403 404 409 410 500 |

Netflix

| Status Codes | 200 201 304 400 401 403 404 412 500 |

Facebook

HTTP Status Code: 200

{"type": "OAuthException","message": "(#803) Some of the aliases you requested do not exist: foo.bar"}

Twilio

HTTP Status Code: 401


Digg

| Status Codes | 200 400 401 403 404 410 500 503 |

SimpleGeo

HTTP Status Code: 401

{"code": 401,"message": "Authentication Required"}
REST Ambiguities (Versioning of APIs)

From Client Application..

GET - http://ex.com/api/v1/user/1  (V1 version of resource)

PUT - http://ex.com/api/v2/user/1  (Update V2 version of resource with V1 representation with client)

{ ....

}

-- Can lead to unpredictable issues (Advised to avoid versioning)
REST Ambiguities ( Non CRUD Operations )

Question :- How do I create an API to GET Validation Rules for a password field for better user experience.

Is password field a resource on my server ? --- NO

Is Validation Rule a resource on my server ? --- NO
When I started ...
Just make it work ...

Hoorah..... !     I have implemented REST

>>>
So..

What do we care?

What are the ways ahead!

Are there any better alternatives?

- Message Transfer Efficiency
- HTTP/2.0 protocol
- Better User Experience
- Microservices
Message Transfer Efficiency

- Widely used format is JSON (ASCII)
- Key and value are part of the message being transmitted
- Gzip can help to an extend

Alternative

- Protocol buffers: from various benchmarks have figured out that we get around
  - 10% efficiency in terms of size and speed over JSON. (From Javascript client to java language)
  - 78% efficiency in terms of size and speed over JSON. (From Java to Java)

-- So can be useful in microservices
HTTP/2.0 Support

- Single TCP Connection
- Gzip (Headers + Data)
- Server Push
- Multiplexing
- Prioritization
- Flow control

Benefit for REST APIs

- Compression of headers & data
- Shorten RTTs, (Single TCP Connection)
- Multiplexing
- Server push may replace polling..
- Default Secure communication (TLS) from browsers.
RPC frameworks

**gRPC**

- Takes advantage of feature set of HTTP/2
- Takes advantage of Protocol Buffers
- IDL to describe service API
- Automatically generates client stubs and abstract server classes in 10+ languages

-- Can be excellent to implement communication between private APIs backed by Microservices.

**Thrift**

- IDL to describe service API
- Automatically generates client stubs
- Modular serialization
- Modular transport mechanism.
## Performance Benchmarks ( JSONRPC Vs gRPC )

Tests send 300,000 requests to key/value stores. One with `jsonrpc`, the other with `gRPC`. Both `jsonrpc` and `gRPC` code use only one TCP connection. And another `gRPC` case with one TCP connection but with multiple clients:

<table>
<thead>
<tr>
<th>RPC</th>
<th># of requests</th>
<th># of clients</th>
<th>total time</th>
<th>per-request time</th>
</tr>
</thead>
<tbody>
<tr>
<td>jsonrpc</td>
<td>300,000</td>
<td>1</td>
<td>8m7.270s</td>
<td>1.624ms</td>
</tr>
<tr>
<td>gRPC</td>
<td>300,000</td>
<td>1</td>
<td>36.715s</td>
<td>122.383µs</td>
</tr>
<tr>
<td>gRPC</td>
<td>300,000</td>
<td>100</td>
<td>7.167s</td>
<td>23.892µs</td>
</tr>
</tbody>
</table>

The Verdict (Personal Opinion)

- When should you use REST?

  I just want to make an API Call, REST Tries to force HTTP Semantics in the API, which makes it confusing for a developer at least.

  Though, REST is not dead. As Chrome + Firefox needs to provide basic infrastructure for RPC frameworks. And REST will be there for some time at least until we get support in browsers.

- When should you use RPC?

  If performance is primary concern, RPC can provide an edge by leveraging serialized data formats and alternative transport mechanisms. And for an engineer .. i think it is.
References

- REST: I don't Think it Means What You Think it Does
- The never-ending REST API design debate
- Is Your REST API RESTful? Pragmatic REST APIs (MUST), Pragmatic REST (MUST)
- Protocol Buffers, Protocol Buffers & JS
- gRPC, How google designs APIs (MUST)
- Performance Evaluation - gRPC IO
- gRPC vs REST?? (MUST)
- gRPC - The story of microservices at square (MUST)
- A general RPC framework that puts mobile and HTTP/2 first
- Take a REST with HTTP/2, Protobufs and Swagger
- Scalable Realtime Microservices with Kubernetes and gRPC
- Java EE Next HTTP 2 and REST Opportunities
- HTTP/2, HTTP/2 is here by google (MUST)
- pRPC. unary gRPC on AppEngine, in browser, with OAuth 2.0
- http://unrestful.io/2015/12/01/devoxx.html
- https://www.api2cart.com/blog/rest-api-services-can-expect-http2/
- Demo : http://http2.golang.org/gophertiles?latency=1000
- Benchmarks : Link 1, Link2, Link3 (MUST), Link4 (MUST)
- Latency delays related to networks
- https://martinfowler.com/articles/richardsonMaturityModel.html
- Art of effective API Design
- https://sookrceff.com/post/apil/swagger-thrift-or-grpc/
- Yesterday's perf best-practices are today's HTTP/2 anti-patterns (MUST)
- Latency delays related to networks
- https://martinfowler.com/articles/richardsonMaturityModel.html
- Art of effective API Design
- https://sookrceff.com/post/apil/swagger-thrift-or-grpc/
- Yesterday's perf best-practices are today's HTTP/2 anti-patterns (MUST)
Icon References

- http://firstwhistle.esy.es/images/benefits.png
- https://d30y9c0d8u7zg0.cloudfront.net/png/138240-200.png
- https://d30y9c0d8u7zg0.cloudfront.net/png/81984-200.png
- http://www.cmsqualityconference.com/images/graduated20-p-500x500.png
- http://imageog.flaticon.com/icons/png/512/38/38823.png?size=1200x630f&pad=10,10,10,10&ext=png&bg=FFFFFF
- Icons made by http://www.flaticon.com/authors/simpleicon from http://www.flaticon.com is licensed by http://creativecommons.org/licenses/by/3.0/ CC 3.0 BY
THANK YOU :) 

Questions :- birju.s@endurance.com