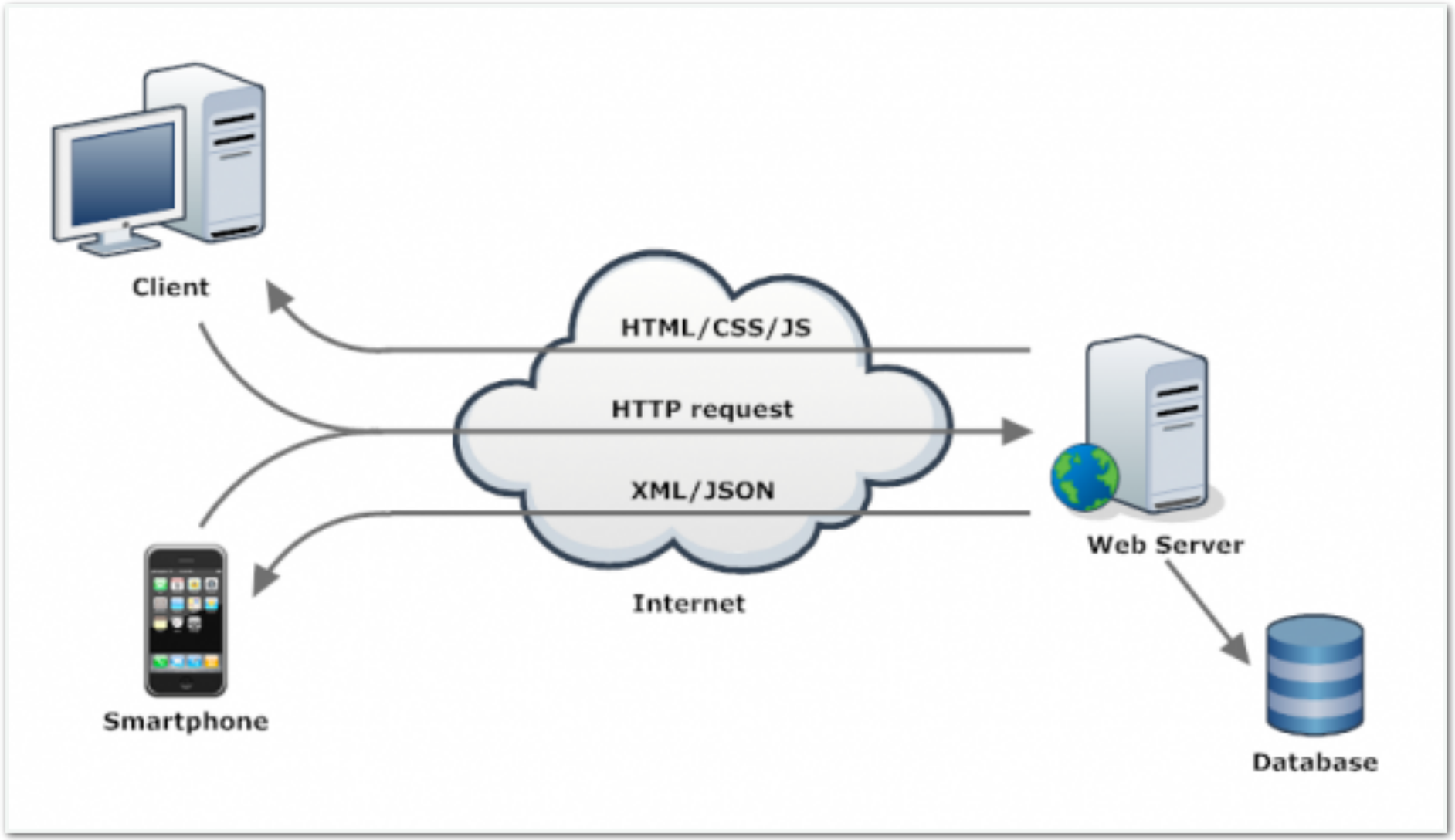
REST

REpresentational State Transfer

????

Web Service


Client
HTML/CSS/JS
HTTP request
XML/JSON
Internet
Web Server
Database
Smartphone

Web Services - How to implement ?

2 approaches

using

1. SOAP protocol

2. REST constraints/patterns/styles

Web Services - How to implement ?

If you are java developer ,

we have 2 standard specifications to implement web services

JAX-WS - SOAP protocol based

JAX-RS - using REST constraints/patterns/styles

+

also using Spring Web & MVC

www

Tim Berners-Lee

the inventor of the World Wide Web

WWW

Tim Berners-Lee had invented and implemented

1. URI
2. HTTP
3. HTML
4. web-server
5. web-browser (www) (Nexus)

web

Web growing too large, too fast, and it is heading toward collapse

Web's traffic outgrowing the capacity of the Internet infrastructure

web

Web's core protocols were not uniformly implemented.

They lacked support for caches and other stabilising intermediaries.

With such rapid expansion, it is unclear if the Web would scale to meet the increasing demand.

Architectural styles

‘Roy Thomas Fielding ‘ in his 2000 PhD dissertation

"*Architectural Styles and the Design of Network-based Software Architectures*”

introduced some styles/patterns/constraints for web’s scalability needs

and he called them as ‘ REST’

Fielding developed the REST architectural style in parallel with HTTP 1.1 of 1996-1999, based on the existing design of HTTP 1.0 of 1996.

Architectural constraints or REST constraints/styles/patterns

1. Client–server
2. Stateless
3. Cacheable
4. Layered system
5. Code on demand (optional)
6. Uniform interface (API)

Referring

Roy Fielding - Presentation on REST

Architectural properties

The architectural properties affected by the constraints of the REST architectural style are:

Performance

Scalability to support large numbers of components and interactions among components.

Simplicity of interfaces

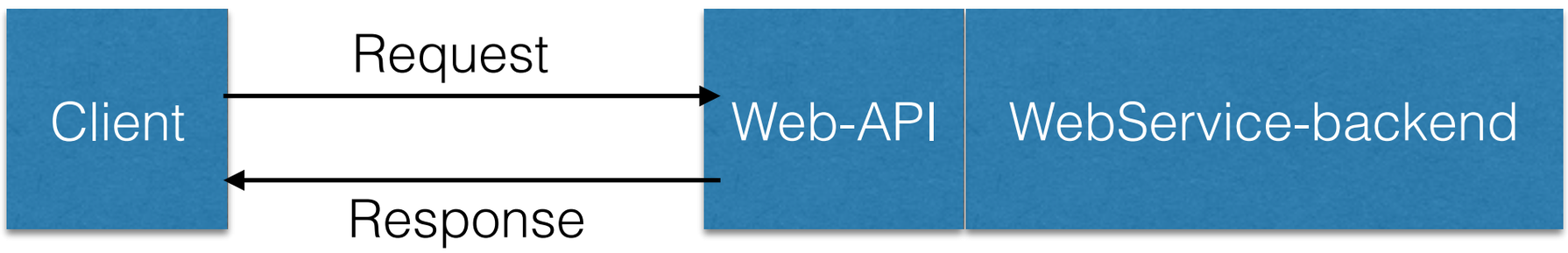
Modifiability of components to meet changing needs

Visibility of communication between components by service agents

Portability of components by moving program code with the data

Reliability is the resistance to failure at the system level in the presence of failures within components, connectors, or data

WebService - Web API



REST API

The REST architectural style is commonly applied to the design of APIs for modern web services.

A Web API conforming to the REST architectural style is a REST API.

Having a REST API makes a web service “RESTful.”

REST API

A REST API consists of an assembly of interlinked resources.

This set of resources is known as the REST API's resource model.

Well-designed REST APIs can attract client developers to use web services.

REST API - design

Applying ‘ Uniform Interface ‘ style

Uniform interface

The four constraints for this uniform interface are:

1. Identification of resources (with URI)
2. Manipulation of resources through these representations
3. Self-descriptive messages
4. Hypermedia As The Engine Of Application State (HATEOAS)

REST API - design

Some best practices for REST API design are implicit in the HTTP while other have emerged over the past few years.

Rules for ‘REST API’ Design

1.Identification of resources

REST APIs use Uniform Resource Identifiers (URIs) to address resources.

REST API designers should create URIs that convey a REST API's resource model to its potential client developers.

URI Format

URI = scheme "://" authority "/" path ["?" query] ["#" fragment]

URI - Rules

Forward slash separator (/) must be used to indicate a hierarchical relationship

http://api.canvas.restapi.org/shapes/polygons/quadrilaterals/squares

A trailing forward slash (/) should not be included in URIs

http://api.canvas.restapi.org/shapes/

http://api.canvas.restapi.org/shapes

URI - Rules

Hyphens (-) should be used to improve the readability of URIs

http://api.example.restapi.org/blogs/mark-masse/entries/this-is-my-first-post

Underscores (_) should not be used in URIs

Lowercase letters should be preferred in URI paths

http://api.example.restapi.org/my-folder/my-doc
http://api.example.restapi.org/My-Folder/my-doc

URI - Rules

File extensions should not be included in URIs

http://api.college.restapi.org/students/3248234/transcripts/2005/fall.json

http://api.college.restapi.org/students/3248234/transcripts/2005/fall?type=json

http://api.college.restapi.org/students/3248234/transcripts/2005/fall

Accept: application/xml or application/json

URI Authority Design - Rules

Consistent sub-domain names should be used for your APIs

http://api.soccer.restapi.org

http://developer.soccer.restapi.org

Resource Modelling

The URI path conveys a REST API's resource model

with each forward slash separated path segment corresponding to a unique resource within the model's hierarchy

http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet

http://api.soccer.restapi.org/leagues/seattle/teams

http://api.soccer.restapi.org/leagues/seattle

http://api.soccer.restapi.org/leagues

http://api.soccer.restapi.org_ root resource

URI Path Design

A singular noun should be used for document names

http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet/players/claudio

A plural noun should be used for collection names

http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet/players

URI Path Design

A plural noun should be used for store names

http://api.music.restapi.org/artists/mikemassedotcom/playlists

A verb or verb phrase should be used for controller names

http://api.college.restapi.org/students/morgan/register

URI Path Design

Variable path segments may be substituted with identity-based values

http://api.soccer.restapi.org/leagues/{leagueId}/teams/{teamId}/players/{playerId}

URI Path Design

CRUD function names should not be used in URIs

GET /deleteUser?id=1234

GET /deleteUser/1234

DELETE /deleteUser/1234

POST /users/1234/delete

URI Query Design

The query component of a URI may be used to filter collections or stores

GET /users

GET /users?role=admin

The query component of a URI should be used to paginate collection or store results

GET /users?pageSize=25&pageStartIndex=50

Interaction Design with HTTP

imp note:

GET and POST must not be used to tunnel other request methods.

HTTP - Methods

GET must be used to retrieve a representation of a resource

HEAD should be used to retrieve response headers

HTTP- Methods

PUT must be used to both insert and update a stored resource

PUT must be used to update mutable resources

HTTP Methods

POST must be used to create a new resource in a collection

POST must be used to execute controllers

HTTP Methods

DELETE must be used to remove a resource from its parent

OPTIONS should be used to retrieve metadata that describes a resource's available interactions

POST vs PUT

```
PUT /article/1234 HTTP/1.1
<article>
    <title>red stapler</title>
    <price currency="eur">12.50</price>
</article>
```

```
POST /articles HTTP/1.1
<article>
    <title>blue stapler</title>
    <price currency="eur">7.50</price>
</article>

HTTP/1.1 201 Created
Location: /articles/63636
```

POST vs PUT

```
PUT /articles/green-stapler HTTP/1.1
<article>
    <title>green stapler</title>
    <price currency="eur">9.95</price>
</article>

HTTP/1.1 201 Created
Location: /articles/green-stapler
```

http://restcookbook.com/HTTP%20Methods/put-vs-post/

HTTP Methods

HTTP Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update	404 (Not Found), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	404 (Not Found), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

Referring

https://developer.mozilla.org/en-US/docs/Web/HTTP

Response Status Codes

1xx: Informational -Communicates transfer protocol-level information.

2xx: Success - Indicates that the client's request was accepted successfully.

3xx: Redirection - Indicates that the client must take some additional action in order to complete their request.

4xx: Client Error - This category of error status codes points the finger at clients.

5xx: Server Error - The server takes responsibility for these error status codes

Referring

https://developer.mozilla.org/en-US/docs/Web/HTTP/Response_codes

Metadata Design- HTTP Headers

Content-Type must be used

Content-Length should be used

Last-Modified should be used in responses

Location must be used to specify the URI of a newly created resource

Cache-Control, Expires, and Date response headers should be used to encourage caching

Cont..

Cache-Control, Expires, and Pragma response headers may be used to discourage caching

Caching should be encouraged

Media Types

Media Type Syntax

type "/" subtype *(";" parameter)

Content-type: text/html; charset=ISO-8859-4

Content-type: text/plain; charset="us-ascii"

Vendor-Specific Media Types

application/vnd.ms-excel

application/vnd.lotus-notes

text/vnd.sun.j2me.app-descripto

Media Type Design

Application-specific media types should be used

Media type negotiation should be supported when multiple representations are available

Media type selection using a query parameter may be supported

Using Accept Header prefered

Representation Design

JSON should be supported for resource representation

JSON must be well-formed

XML and other formats may optionally be used for resource representation

Hypermedia Representation

A consistent form should be used to represent links

A consistent form should be used to represent link relations

Hypermedia Representation

refer

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