

REST Architecture and RESTful APIs

Lecture 12: CPEN 400A
Based on CS498RK taught at UIUC (used with permission), and Roy Fielding's PhD thesis at UC Irvine

Outline

- What is REST ?
- HTTP and REST
- RestFul APIs
 - No code samples, theory I→ design ref sl.

REST

- Stands for Representational State Transfer
- Proposed by Roy Fielding at UC Irvine as part of his PhD dissertation
 - Already implemented in Apache web server by Roy → Already in production . *fundamental principles of implement.*
 - Basis for much of the modern web and its design
 - Original definition has been significantly extended

- 90s, JS just developed ⇒ borrowed to architectural pattern in use today.

So what's this REST thing ?

- REST is what you've been doing already in web applications. Example: accessing a URL
 - It's an architectural style, NOT a standard */protocol.*
 - Set of design principles and constraints that characterize web applications
- But REST is a general principle that goes beyond Web (web is one implementation)
 - design for mobile app / IoT applications · by REST, very suitable for web

Why REST ?

→ other: SOAP .
- anti-REST .

- Performance ①
- Scalability ② → massive
- Simplicity of interfaces ③ → easy to interf.
- 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 or the resistance to failure at the system level
 - you can engineer it to be more fault tolerant

The six principles of REST

- Client-Server
 - Statelessness
 - Cacheable
 - Layered System
 - **Uniform Interface (this is the most important)**
 - Code on Demand (Optional)
- Architectural Standards
- In general,
restful API
will follow
- 

Client-Server

- Clear separation between clients and servers
- Servers and clients can be replaced and developed independently as long as the interface between them is not altered



Keep interface compatibility between multiple components.

Stateless

- Server doesn't know about client's application state – passed in by client
- Server is replaceable and can pass session state to another server or database
 - use any server -
✓
- Pass representations around to change state
 - Representation must contain all the needed info

Cacheable

- Caching improves performance, but can compromise on freshness
- Responses are assumed to be cacheable by default
- If response does not wish to be cached, it must explicitly mark itself as such
 - eliminating perf bottleneck
 - Temporary failures

Layered System

- Client should not be able to tell if it is directly connected to server or through an intermediary (e.g., proxy, firewall etc)
 -
- Allows scalability, e.g., through load balancing
- Security policies may be applied at proxy
 - Not always true (e.g. HTTPS)

Uniform Interface

- Identification of resources
- Manipulation of resources through these representations → e.g. HTTP requests (GET, POST)
- Self-descriptive messages
- **hypermedia** as the engine of application state (HATEOAS)
 - ↳ everything communicated in hypermedia → uniform interface

Code on Demand

pretty good.

- This is the only optional principle
 - Extend functionality of client by transferring logic to the client side
 - Examples are JavaScript code, Java Applets
- Because in 1990s, mostly HTML -

Outline

- What is REST ?
- HTTP and REST
- RestFul APIs

HTTP

Hypertext Transfer Protocol

request-response protocol

“all about applying verbs to nouns”

nouns: resources (i.e., concepts)

verbs: GET, POST, PUT, DELETE

give

↓
Add / modify

↓
Do something
to resource

↑
remove

APPLY
to.

Resources

If your users might “want to create a hypertext link to it, make or refute assertions about it, retrieve or cache a representation of it, include all or part of it by reference into another representation, annotate it, or perform other operations on it”, make it a resource

can be anything: a document, a row in a database, the result of running an algorithm, etc.

- do not worry abt specifics www.w3.org/TR/2004/REC-webarch-20041215/
of resources
- Uniform way to access resource] → Uniform principle

URL

Uniform Resource Locator

every resource must have a URL

type of URI (Identifier)

↑ generalized -

specifies the location of a resource on a network

- REST - every resource should have URL
- unique way to address resource -

REPRESENTATION OF RESOURCES

when a client issues a GET request for a resource,
server responds with **representations** of resources
and not the resources themselves

→ COPY, distinct from original
resource. ↗ / representations

any machine-readable document containing any
information about a resource

server may send data from its database as HTML,
XML, JSON, etc.

web.archive.org/web/20130116005443/http://tomayko.com/writings/rest-to-my-wife

representation of resource .

Representational State Transfer

- Representations are transferred back and forth from client and server
- Server sends a representation describing the state of a resource
- Client sends a representation describing the state it would like the resource to have

Multiple Representations

- A resource can have more than one representation: different languages, different formats (HTML, XML, JSON)
- Client can distinguish between representations based on the value of Content-Type (HTTP header) *→ client knows*
- A resource can have multiple representations—one URL for every representation

*→ No implicit way to convert.
- get from new URL.*

Http Methods

- Get/Head
- Delete
- Post
- Put
- Patch
- Options

GET and Head Methods

↳ Quickly check what is the resource.

- Retrieve representations of resources
- No side effects: not intended to change any resource state
- No data in request body
- Response codes: 200 (OK), 302 (Moved Permanently), 404 (Not Found)
- Safe method (i.e., does not modify any resources)
- Idempotent (called many times, same result **on the server side – in this case no result**) → trivial.

- if server wants to cache, doesn't matter = - client view do not change -
- client side visible state.

Delete Method

- Destroy a resource on the server
- Success response codes: 200 (OK), 204 (No Content), 202 (Accepted)
 - Why is this important ?
→ delete same resource everytime
 - Can return 404 second time to indicate error
→ ensure that resource is deleted.
→ some fix on server.
- Not safe, but idempotent (i.e., can be called many times but will have same result on the server side – need not return the same value)
 - e.g. if network problem while del.
 - just del again, result unchanged
 - reason about multiple req -

Swiss Army Knife. ↗

Post Request

- Upload data from the browser to server
 - Usually means “create a new resource,” but can be used to convey *any* kind of change: PUT, DELETE, etc.
 - Side effects are likely ↘
- Data contained in request body
- Success response codes:
 - 201 (Created): **Location** header contains URL for created resource;
 - 202 (Accepted): new resource will be created in the future
- Neither safe nor idempotent

Modify
server ↘

→ will make stuff twice.
- not same result -
Careful: if failure, cannot reason state.

Put Method

- Request to modify resource state
 - Success response codes:
 - 200 (OK)
 - 204 (No Content)
 - 201 (Created) – see below
 - Not safe, but idempotent (why ?)
 - Can also be used like POST idempotent
 - Will create the resource if it does not exist
 - URI can be chosen by the client (may be risky)
 - Not widely used in practice
 - For a particular resource.
 - Create at most 1 time
- Modifies*
- ensure that val. will be set to val.*
- makes only once*
- good!*
- in style guide.*
- \Rightarrow
- use PUT instead of POST*
- mostly modify or create*
- if does not exist*

Patch Method

- Representations can be big: PUTs can be inefficient
- Send the server the parts of the document you want to change → Δ only
- Neither safe nor idempotent

↓
Making
change

→ Delta or Delta is
not the same.

e.g. $x++ \Rightarrow$,

Outline

- What is REST ?
- HTTP and REST
- **RestFul APIs**

Web API

- Application program interface (API) to a defined request-response message system between clients and servers
 - Accessible via standard HTTP methods
 - e.g. AJAX, Assts, etc.
- Request URLs that transfer representations (JSON, XML)
 - Conform REST principles

Restful APIs: Features

- Application program interface to a defined request-response message system between clients and servers
- Accessible via standard HTTP methods
- Request URLs that transfer representations (JSON, XML)

Designing Restful APIs

Apply Verbs to Nouns

Http Methods

Resources



Collections

<VERB> `http://example.com/users`

list of all user.

GET Return all the objects in the collection

POST Create a new entry in the collection;
automatically assign new URI and return it

- Add new entry + URI / URL

PUT and **DELETE** not generally used

- Doesn't make sense to modify whole coll.

Elements

Whatever conversion
you wish.

VERB> `http://example.com/users/1234`

GET Return the specific object in collection

PUT Replace object with another one

e.g. change age, specify fields. w/ new obj

DELETE Delete element

X **POST** not generally used

- not creating new entry

Using Parameters for Queries

```
<VERB> http://example.com/users/12345?  
where={"num_posts": {"$gt":100}}}
```

Json-encoded filter

other parameters can be used to select fields, sort, etc.

parameters can also be URL-encoded

Example: Pagination

- if too many users

GET `http://example.com/users?`

offset=60&limit=20

↳ Start from 60th, up to 20 users.

offset *ith object & limit*

limit *number of returned objects*

can also use **Link** header to specify next,
prev, first, last URLs

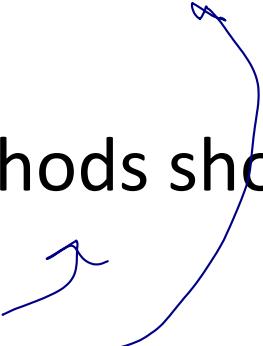
CheckList: Restful APIs

- Use nouns but no verbs. Always Use plural nouns.
-
- Don't expose irrelevant nouns
 - Basic minimum, less maint.
- GET method and query parameters should not alter the state (safe)

→ users,
pages,
jobs etc.

- PUT and DELETE methods should be idempotent

- Do not change semantics.
- Respect rules of GET, PUT, DELETE.



Class Activity

- Design a simple REST API to perform the following actions in a Phonebook application
 - Retrieve the list of all contacts in the phonebook
 - Retrieve a specific contact given their key
 - Retrieve the info of a specific contact given their first name and last name
 - Add a new contact to the phonebook
 - Modify the details of an existing contact
 - Remove a contact from the phonebook

Solution to the Activity - Retrieval

- Use nouns rather than verbs
 - To request all contacts, use
 - GET foo.com/contacts
 - To request a specific contact given a key, use
 - GET foo.com/contacts/12345
 - To find a contact (by first-name and last name),
 - GET foo.com/contacts?fname="ABC"&lname="XYZ"
 - To paginate the contacts,
 - GET foo.com/contacts?offset=100&limit=20

Solution to the Activity - Add

- Add should be a POST request as it modifies the state of contacts, and is not idempotent

POST foo.com/Contacts/

Send contact details in the body of the request, as JSON formatted object (say)

NOTE: We Post on the collection Contacts

Solution to the Activity - Modify

- Can use PUT if key is known (better than POST as it's idempotent). Can also use PATCH for partial updates to save bandwidth, if needed.

PUT foo.com/Contacts/12345

Send the new data (to be modified) in the body of the PUT request – assumes key is present

Solution to the Activity – Delete

- Use Delete method in HTTP to remove the object given its key (idempotent). Should not do anything if contact is not present in server.

DELETE foo.com/contacts/12345

can also be used for multiple contacts as follows

DELETE foo.com/contacts?firstName=“Jack”

Outline

- What is REST ?
- HTTP and REST
- RestFul APIs