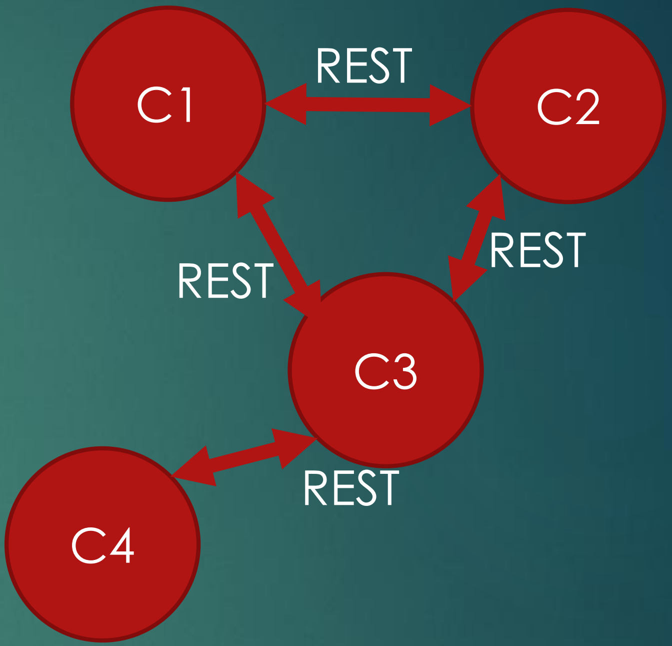
REST

K V SUBRAMANIAM

Introduction: Why Study REST?

- Distributed systems have multiple components
- Frequently use REST for communication



References

- https://www.infoq.com/articles/rest-introduction/
- http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

REST

SOURCE: LEARN REST IN 18 SLIDES, SURAJ GUPTA, OBEAUTIFULCODE.COM, 2014

Why Learn Rest

BUILD **CLIENT-FRIENDLY** DISTRIBUTED SYSTEMS

SIMPLE TO UNDERSTAND

EASY TO SCALE

What is REST?

Is it a standard?NO

- REST is a programming style

Example:

HTTP can also be used in a non-RESTful way

- Example: SOAP services that use http to transport data

Familiarity Check

- What is
  - URLs, URIs
  - Hypertext
  - Accept: text/html
  - 200OK, 404 not found
  - GET/PUT/POST

URI- naming an object

Hypertext – data with meta-data

Format of data

Status

Operations

5 major concepts of REST

- Resources
- Representations
- Operations
- Hypertext
- Statelessness

Resources

- Resource is a **“Thing”**
- You need to give an identifier for a “thing”
- Take for example “light on board in the seminar hall”
- If you need to control it, you need to name it.
- For example
  - /vidhansoudha/groundfloor/cmoffice /light/1
    - Refers to the first light on the room
    - Each light can be separately controlled.

URLs and URIs

URI – Uniform Resource Identifier

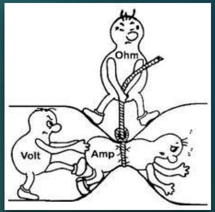
- The name of the object on the web
- Identifies a resource by
  - Name
  - Location
  - Or both

URL – Uniform Resource Locator

- Subset of an URI
- Specifies where to find a resource – the location
- How to retrieve the resource

REST design principle – Identify everything that is worth being identified.

Representations



- How does a client know what to do
  - with data it receives
  - On fetching a URL
- RESTful systems → empowers client to ask for data in a form they understand
- Type/Form/Representation – MIME types
  - Over 1000 standard MIME types
- Can request in XML, JSON etc..
- Example
  - GET /pages/archive HTTP/1.1
  - Host: obeautifulcode.com
  - Accept: text/html

Class Exercise (5 mins)

Consider a web application for building a BookShop using a web application

Your shop should support

- Retrieve book details like price, recommendations
- Adding a book to a shopping cart
- Like a book
- Sending recommendation of a book to a friend
- Deleting a recommendation for a book.

List out the resources for your application and the representations if you would like to see the data in JSON format

Solution

Resources

Books

Recommendations

Shopping Cart

Friend

Representation

Accept: text/JSON

Representations..

- Can we use different URLs for different representations of the same resource
  - https://bblock/seminarhall/board/light1.xml
  - https://bblock/seminarhall/board/light1.html
- Not required. Use a different representation for same resource.
- If server does not support requested MIME type
  - Return standard error (HTTP 406)
- Using representations with negotiation allows for flexibility

Operations

- When we develop applications, we think of operations
- For BookKarts this could be
  - GetListofBooks()
  - AddBookToShoppingCart()
- Need to define these operations
- No standard style exists
- For functionality and side-effects – consult the manual

Operations - REST

GET

Retrieve representation of a resource

https://vidhansoudha/groundfloor/cmoffice/light1.xml

To check if light is ON

PUT

Create or update resource by replacement.

https://ipl/2026/match17/venue.xml

Create the venue item for a match.

POST

Create or partial update of a resource

https://vidhansoudha/groundfloor/cmoffice/light1.xml

To turn on/off the light

Put ON in the body of the message

DELETE

Remove a resource

https://bcci/2025/contracts/players/viratkohli

Operations: Safe vs Idempotent

Safe

Does not modify the resources

Example

- GET

Idempotent

Idempotent has no additional effect if it is called more than once with ***same input parameters***

Can repeatedly perform operations.

No effect on servers

- How would you like to pay for a seat multiple times?

Operations: Rules

Operation	Safe	Idempotent	When to use
GET	Yes	Yes	Mostly for retrieving resources. Can call multiple times.
PUT	No	Yes	Modifies a resource but no additional impact if called multiple times
POST	No	No	Modifies resources, multiple calls will cause additional effect if called with same parameter
DELETE	No	Yes	Removing a resource.
PATCH	No	Depends	Replaces a specific field in the object. Idempotency depends on implementation. If you update timestamp field, then it changes.

Class Exercise

- For each of the operations, select the REST operation to use for designing the Web APP

Application Operation	REST Operation	Justification
Retrieve book details		
Adding a book to a shopping cart		
Like a book		
Deleting a recommendation for a book.		

Class Exercise - solution

- For each of the operations, select the REST operation to use for designing the Web APP

Application Operation	**REST Operation**	**Justification**
Retrieve book details	GET	Just need to details; safe
Adding a book to a shopping cart	POST	Each invocation will result in another copy of the book in the shopping cart. Non-idempotent.
Like a book	PUT	Does not matter if executed multiple times updates on idempotent
Deleting a recommendation for a book.	DELETE	Remove the resource

Hypertext

- Data returned for a resource
- Can contain embedded links
  - Application can follow these links
  - Key point: State of server is transferred to the client using hyperlinks
- Upto client to follow hyperlinks.

- Example

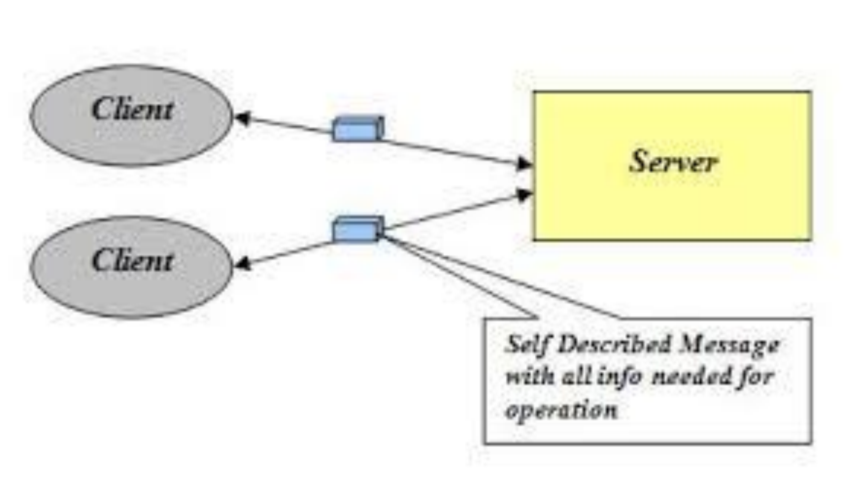
```
<book self=https://bookkarts.com/TheChamberOfSecrets.xml
  <review ref=“http://bookkarts.com/TheChamberOfSecrets/Reviews.xml”>
  <price>INR 500</price>
</book>
```

What is application state?

- For many applications to function, state needs to be maintained across requests
- Example
  1. Login to a web-app like Book Karts
  2. Buy a book
- Somehow the application has to keep track of
  - The **user** who has made the request to buy the book
  - The **user** has now **authenticated** to the app
- Who should keep track of this?
  - Client?
  - Server?

Statelessness

- REST mandates
  - State be turned into resource state or
  - Client takes care of state
- Server **will not** maintain any communication state for a client
- Each client request is treated independently

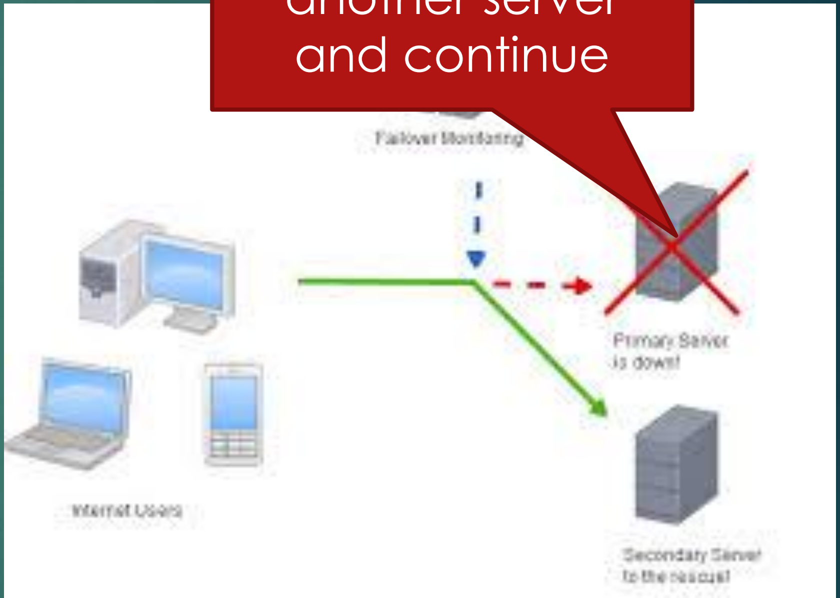


Statelessness - Benefits

Clients isolated against changes on server

Promotes redundancy - unlocks performance

- don't really need to know which server client was talking to
- No synchronization overhead



No state saved on server, so even if server fails, the client connects to another server and continue

Errors

- HTTP response codes are well defined
- Status codes grouped into categories
  - 2xx means action requested was received, understood and processed successfully
  - Body of response can have details of errors
  - Upto client on how errors are handled

The reality of REST

- Resource state implementation is upto the programmer
- If a system requires many resources then REST is probably not a good choice
- Not good choice for real-time or bandwidth constrained scenario due to large number of messages exchanged

Limitations of REST

- Built on HTTP 1.1
  - Lack of multiplexing
    - Head of line blocking
      - Pipelined, but still has to wait for older request
- No streaming support
- API Versioning
- Using JSON has performance problems
- Lack of typing

Further reading